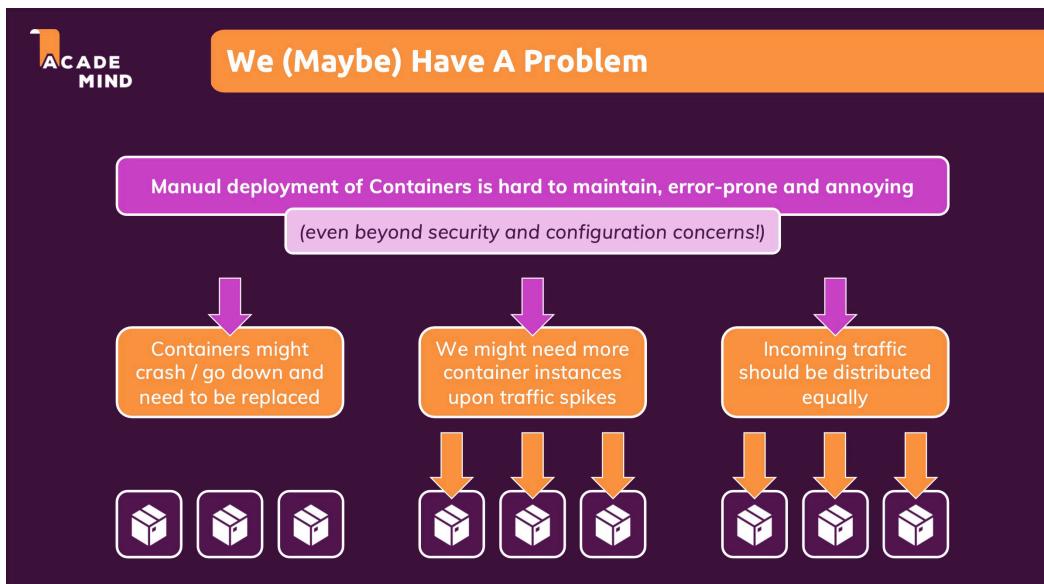


Kubernetes Course:

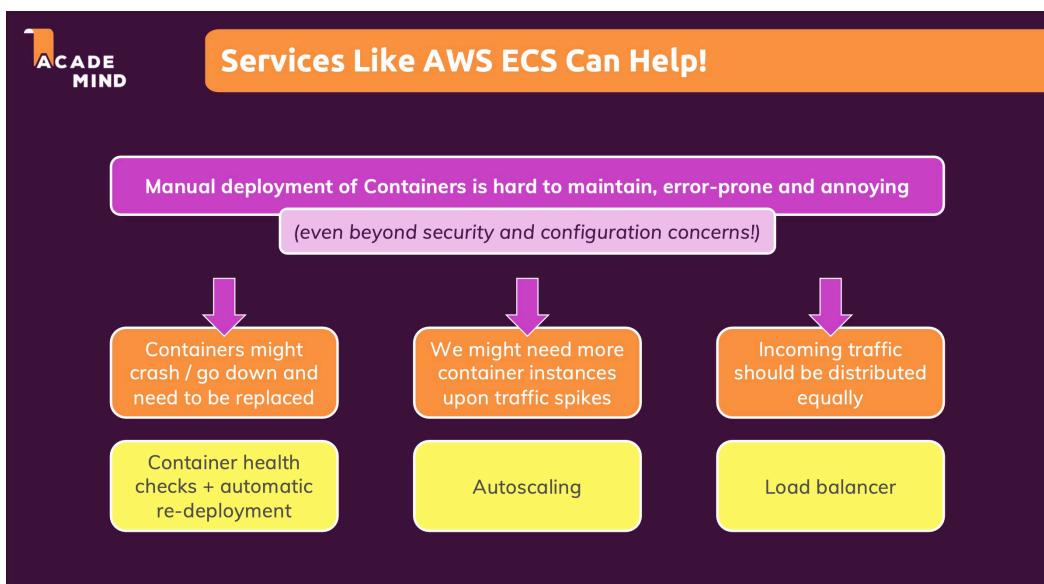
Section 11: Getting Started with Kubernetes:

More Problems with Manual Deployment:

Kubernetes, also known as K8s, is an open-source system for automating deployment, scaling, and management of containerized applications.



Why Kubernetes?:



But That Locks Us In!

Using a specific cloud service locks us into that service

Of course, you might be fine with sticking to one provider though!

You need to learn about the specifics, services and config options of another provider if you want to switch

Just knowing Docker isn't enough!

What Is Kubernetes Exactly?:

Kubernetes To The Rescue



חלוקת תפקדים של קונטAINERים

Kubernetes

An open-source system (and de-facto standard) for orchestrating container deployments

Automatic Deployment

Scaling & Load Balancing

Management

Why Kubernetes?

Kubernetes Configuration

(i.e. desired architecture – number of running containers etc.)

Some Provider-specific Setup or Tool

Any Cloud Provider

or

Remote Machines (e.g. could also be your own datacenter)

Kubernetes allows us to write down a configuration file and, then pass the

configuration file with certain tools to any cloud provider or any machine owned by us.

Which can then pick up the setup we want.

We have one way to setup a configuration file and, this configuration will work on any cloud provider.

The screenshot shows a portion of a Kubernetes Service YAML configuration. The relevant part is:

```
apiVersion: v1
kind: Service
metadata:
  name: auth-service
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-access-log-enabled: "true"
spec:
  selector:
    app: auth-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
  type: LoadBalancer
...
```

Annotations include `service.beta.kubernetes.io/aws-load-balancer-access-log-enabled: "true"`. A callout box highlights this annotation with the text: "Standardized way of describing the to-be-created and to-be-managed resources of the Kubernetes Cluster". Another callout box highlights the `spec.selector` section with the text: "Cloud-provider-specific settings can be added".

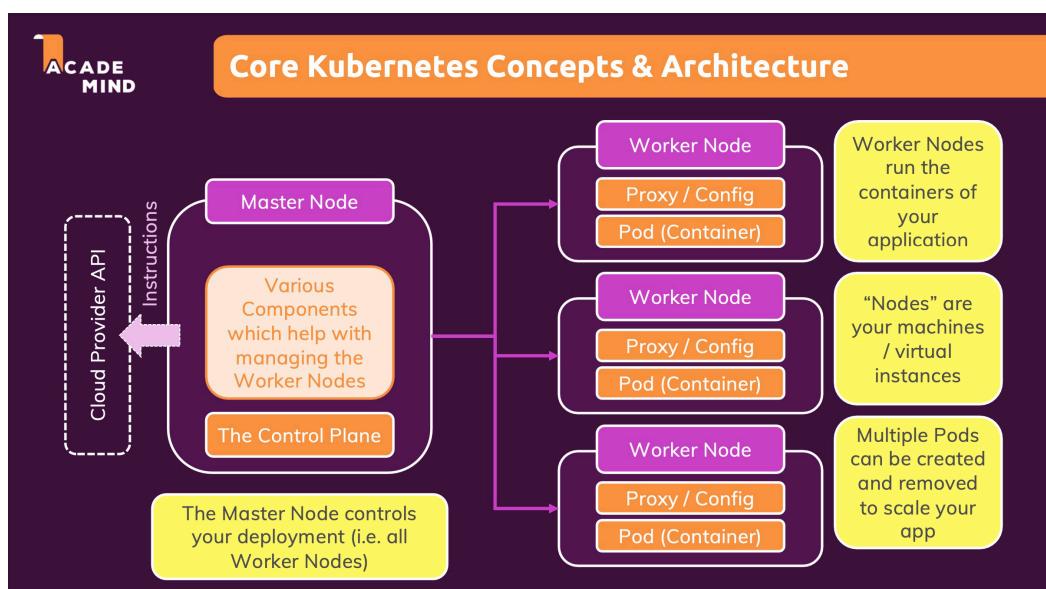
The idea behind Kubernetes - **Having a standardize way of deployment.**

The diagram is titled "What Kubernetes IS and IS NOT". It consists of two columns of three items each, comparing Kubernetes to various concepts.

What Kubernetes IS	What Kubernetes IS NOT
It's not a cloud service provider It's an open-source project	It's not a service by a cloud service provider It can be used with any provider
It's not restricted to any specific (cloud) service provider It can be used with any provider	It's not just a software you run on some machine It's a collection of concepts and tools
It's not an alternative to Docker It works with (Docker) containers	It's not a paid service It's a free open-source project

Kubernetes is like Docker-Compose for multiple machines

Kubernetes: Architecture & Core Concepts:

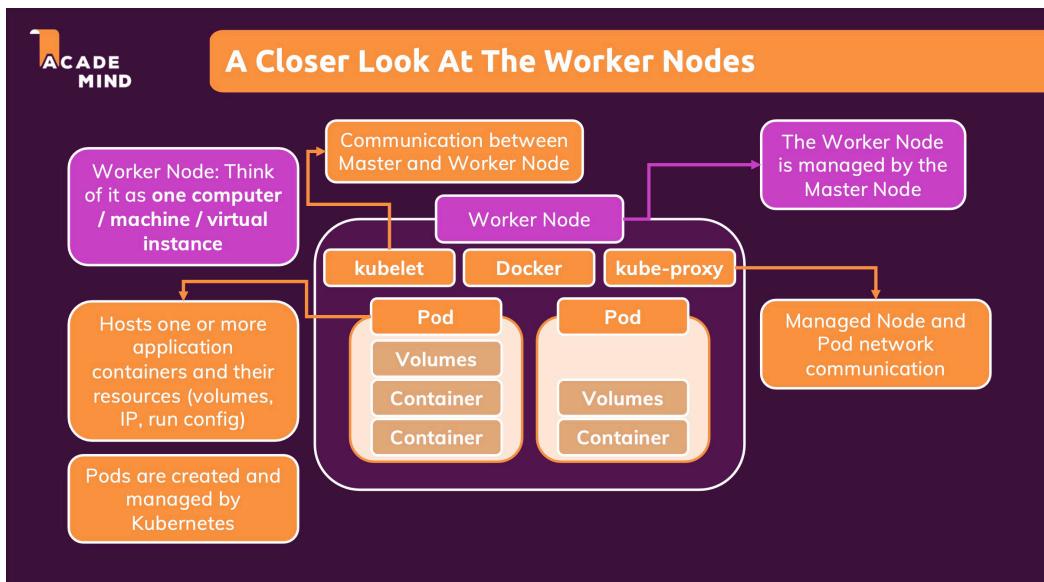


Kubernetes will NOT manage your Infrastructure!:

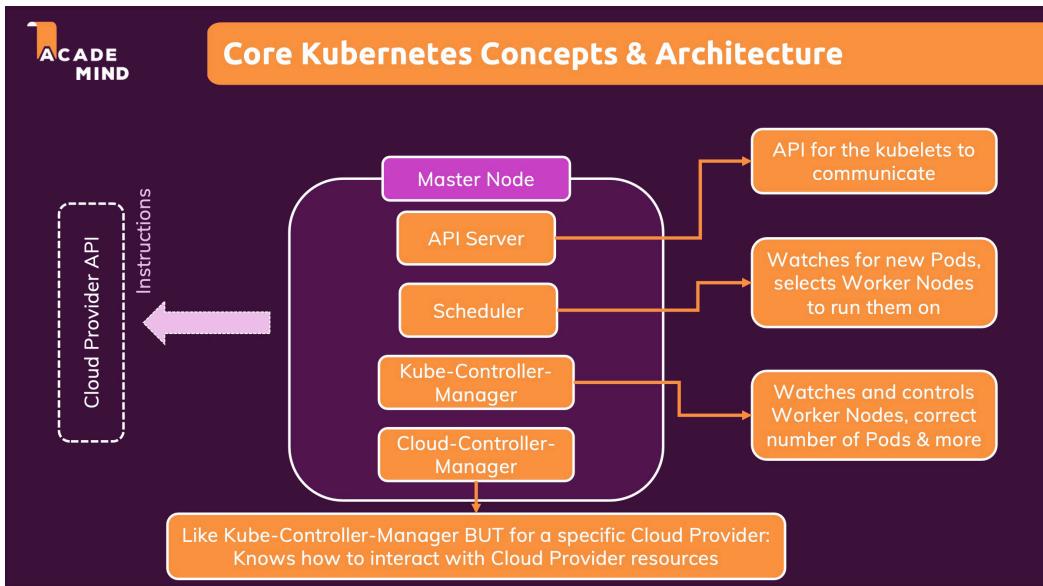
Your Work / Kubernetes' Work



A Closer Look at the Worker Nodes:



A Closer Look at the Master Node:



Important Terms & Concepts:

Core Components	
Cluster	A set of Node machines which are running the Containerized Application (Worker Nodes) or control other Nodes (Master Node)
Nodes	Physical or virtual machine with a certain hardware capacity which hosts one or multiple Pods and communicates with the Cluster
Master Node	Cluster Control Plane , managing the Pods across Worker Nodes
Worker Node	Hosts Pods , running App Containers (+ resources)
Pods	Pods hold the actual running App Containers + their required resources (e.g. volumes).
Containers	Normal (Docker) Containers
Services	A logical set (group) of Pods with a unique, Pod- and Container-independent IP address

Section 12: Kubernetes in Action - Diving into the Core Concepts:

Kubernetes: Required Setup & Installation Steps:

In order to work with Kubernetes we first we need to install it on the machine we want to use it.

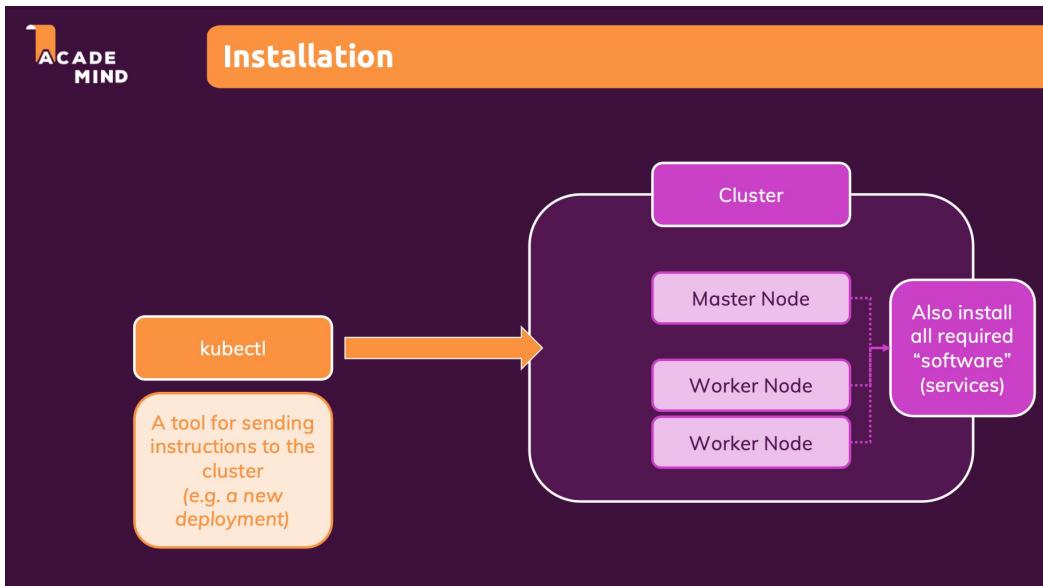
"Kubectl (kube control)" - is a tool to send instructions to the cluster.

We as a developers use the Kubectl to send instructions to the master node - telling the master node what to do with worker nodes.

Minikube installation setup - <https://minikube.sigs.k8s.io/docs/start/> .

Minikube is a tool we install on our local machine in order to setup a dummy cluster that we can work on and test (its another machine - virtual machine).

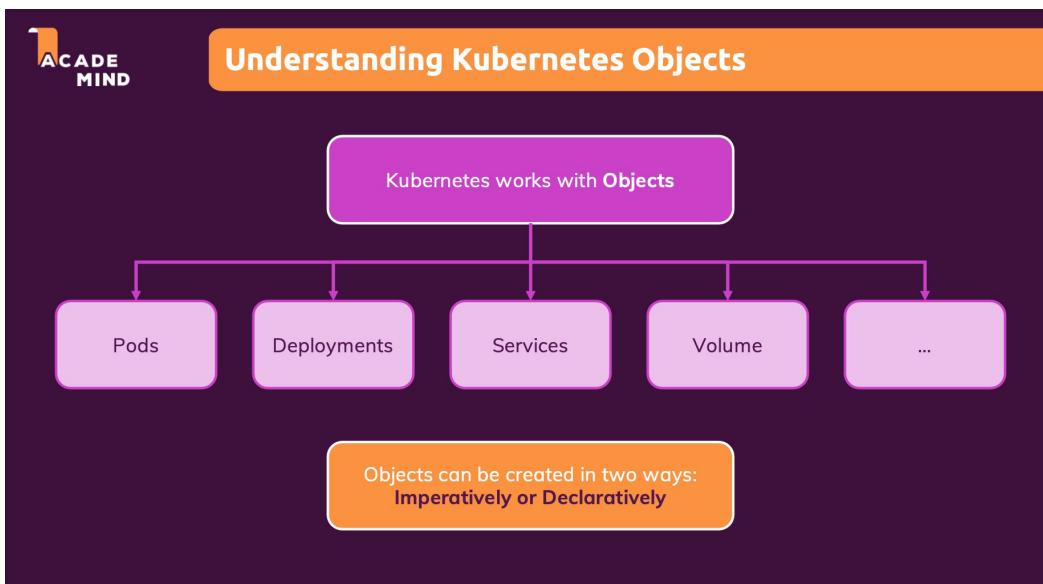
Kubectl installation setup - <https://kubernetes.io/docs/tasks/tools/> .



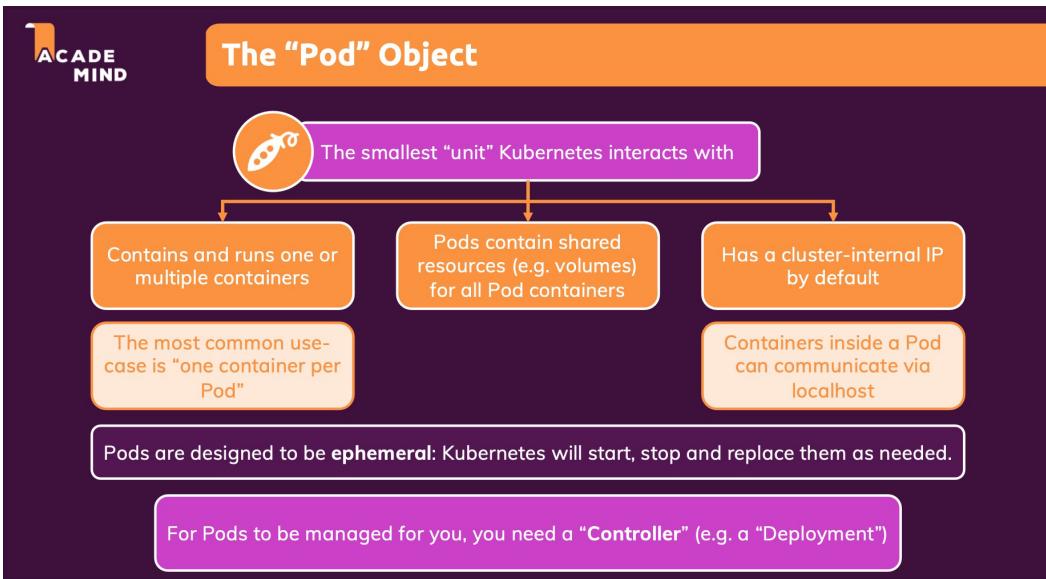
macOS Setup:

<https://www.udemy.com/course/docker-kubernetes-the-practical-guide/learn/lecture/22627609#overview>

Understanding Kubernetes Objects (Resources):



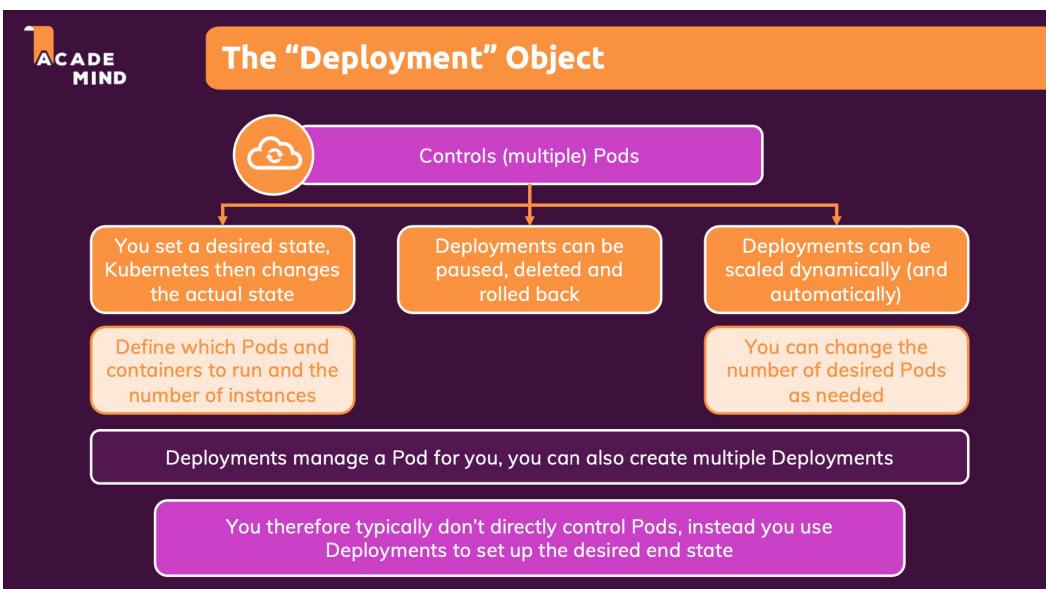
The "Pod" object:



The “Deployment” Object (Resource):

The main object we will work with.

The deployment object is able to control one or more pods.



A First Deployment - Using the Imperative Approach:

- The first step is to create the image of our application (we want to deploy).
- The second step is to send this image to the cluster by creating a deployment object.

To achieve this we use the command – “`kubectl create deployment [OBJECT_NAME] —image=[IMAGE_NAME]`”.

With the “—image” option we specify which image should be used for the container of the pod created in this deployment.

The image should be available in docker hub (the cluster will look for the image in docker hub).

By default this object is sent to the Kubernetes cluster.

Example: “**kubectl create deployment first-app --image=remez19/kub-first-app**”

- With “minikube dashboard” we can see the cluster.

The screenshot shows the Kubernetes Dashboard interface at `127.0.0.1`. The top navigation bar includes a back arrow, a search bar, and tabs for `default` and `Kubernetes Dashboard`. The main header is `Workloads`.

Workload Status: This section displays three large green circles representing the status of Deployments, Pods, and Replica Sets, each labeled "Running: 1".

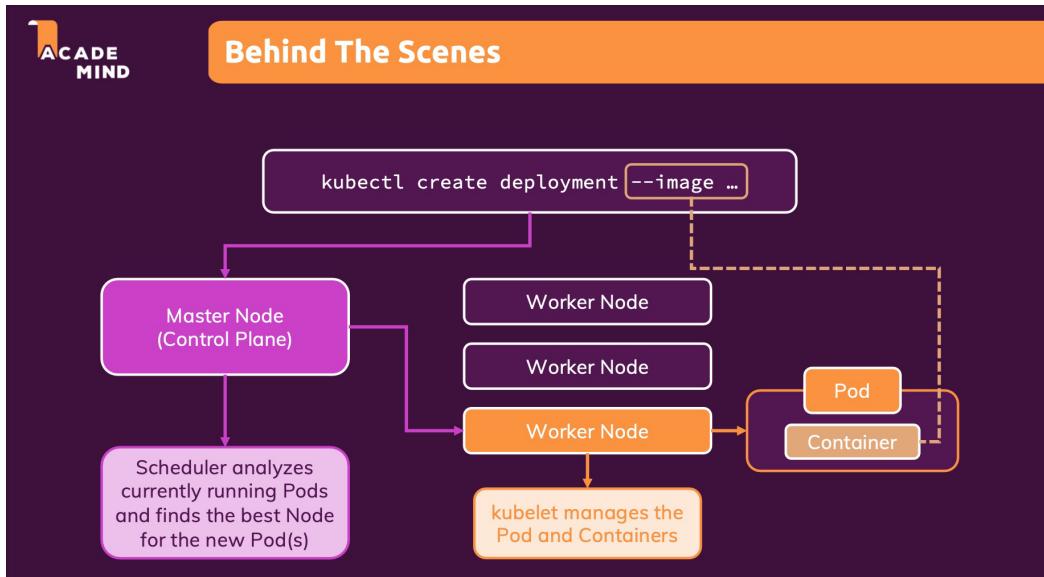
Deployments: A detailed view of the `first-app` deployment is shown, listing its Name, Images, Labels, and Pods. The deployment has one pod running, which is associated with the `remez19/kub-first-app` image and the label `app: first-app`.

Pods: A table lists the current state of pods, showing 1 / 1 pod running.

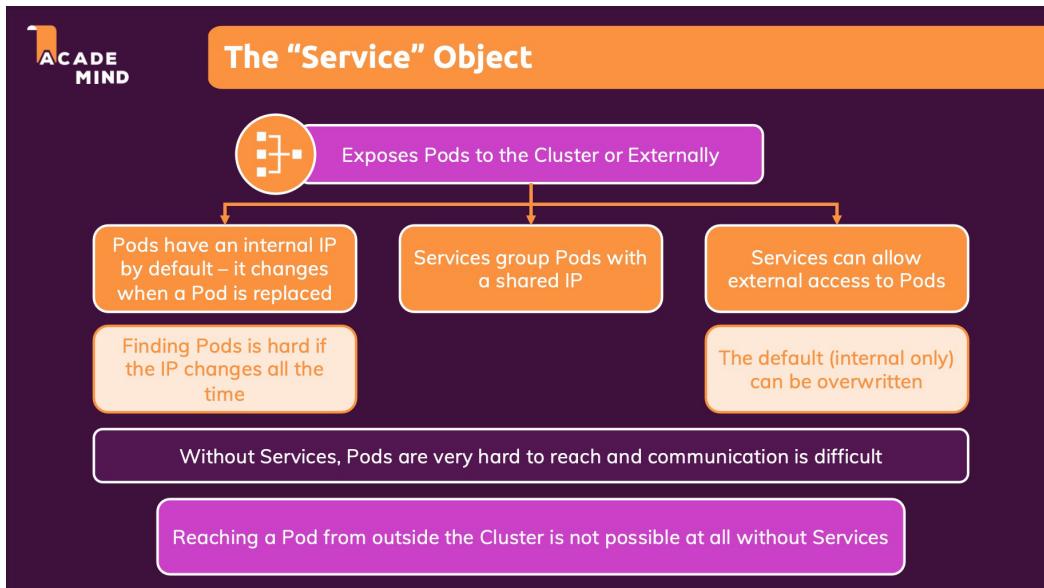
Navigation Sidebar: On the left, a sidebar lists various Kubernetes resources: Cron Jobs, Daemon Sets, Deployments, Jobs, Pods, Replica Sets, Replication Controllers, Stateful Sets, Ingresses, Ingress Classes, Services, Config Maps, Persistent Volume Claims, Secrets, Storage Classes, Cluster Role Bindings, Cluster Roles, Events, Namespaces, Network Policies, and Nodes.

kubectl: Behind The Scenes:

Behind The Scenes



The "Service" Object (Resource):



Exposing a Deployment with a Service:

- The "kubectl expose" command exposes a pod created by deployment by **creating a service**.
- **Example:** "kubectl expose deployment first-app --type=LoadBalancer --port=8080".
- We can see that this service is created by "kubectl get services".
- To get the ip of the pod (see our running container): "kubectl service [SERVICE_NAME]".
- **Example:**

```

kubernetes   ClusterIP   10.96.0.1      <none>        443/TCP    73m
○ remez19  kub-action-01-starting-setup %minikube service first-app
|-----+-----+-----+-----+
| NAMESPACE | NAME | TARGET PORT | URL |
|-----+-----+-----+-----+
| default | first-app | 8080 | http://192.168.49.2:31241 |
|-----+-----+-----+-----+
└ Starting tunnel for service first-app.
|-----+-----+-----+-----+
| NAMESPACE | NAME | TARGET PORT | URL |
|-----+-----+-----+-----+
| default | first-app |          | http://127.0.0.1:53159 |
|-----+-----+-----+-----+
💡 Opening service default/first-app in default browser...
❗ Because you are using a Docker driver on darwin, the terminal needs to be open to run it.

```

Restarting Containers:

Because we used the deployment object we get a behavior - if the container crashed and therefore the pod crash (in which this container is running) then the pod will automatically restart therefore the container will restart.

Scaling in Action:

"**kubectl scale deployment/first-app --replicas=3**" - will create 3 pods (of the same image).

"**kubectl scale deployment/first-app --replicas=1**" - will scale down (1 pod).

Updating Deployments:

- In order to update a deployment we first need to **rebuild our image and give it a different tag**.
- Second step is to **push the image to docker hub**.
- Then we can use: "kubectl set image deployment/[DEPLOYMENT_NAME] [CONTAINER_NAME]=[IMAGE_NAME_DOCKER_HUB]"
- **Example: "kubectl set image deployment/first-app kub-first-app=remez19/kub-first-app:2"**
- We can see the actions in the "Events" tab inside the Kubernetes dashboard.

Events						
Name	Reason	Message	Source	Sub-object	Count	
first-app-5f99f8d677-mplv9.171d492850bf8e	Pulled	Successfully pulled image "remez19/kub-first-app:2" in 4.023206086s	kubelet minikube	spec.containers{kub-first-app}	1	
first-app-5f99f8d677-mplv9.171d49285266e	Created	Created container kub-first-app	kubelet minikube	spec.containers{kub-first-app}	1	
first-app-5f99f8d677-mplv9.171d4928568f09	Started	Started container kub-first-app	kubelet minikube	spec.containers{kub-first-app}	1	
first-app-5f99f8d677-mplv9.171d492743f7f9	Scheduled	Successfully assigned default/first-app-5f99f8d677-mplv9 to minikube	default-scheduler	-	1	
first-app-5f99f8d677-mplv9.171d492760f220	Pulling	Pulling image "remez19/kub-first-app:2"	kubelet minikube	spec.containers{kub-first-app}	1	

Deployment Rollbacks & History:

In cases where we don't want Kubernetes to continue with an update we can use rollback.

"**Kubectl rollout undo deployment/[DEPLOYMENT_NAME]**" - will undo the latest deployment.

If want to go to an older deployment (not the latest one), we can first see our deployment history: "**kubectl rollout history deployment/[DEPLOYMENT_NAME]**".

Than we can get detaild about a specific deployment by using the "REVISION" like: "**kubectl rollout history deployment/[DEPLOYMENT_NAME] — REVISION=[REVISION_NUM]**"

Now to rollout to a specific deployment we can use the revision as such:

"**Kubectl rollout undo deployment/[DEPLOYMENT_NAME] —to-revision=[REVISION_NUM]**".

The Imperative vs The Declarative Approach:

Kubernetes allows us to create resource definition files.



```

ACADE MIND
A Resource Definition

apiVersion: apps/v1
kind: Deployment
metadata:
  name: second-app
spec:
  selector:
    matchLabels:
      app: second-dummy
  replicas: 1
  template:
    metadata:
      labels:
        app: second-dummy
    spec:
      containers:
        - name: second-node
          image: "academind/kub-first-app"

```

Imperative vs Declarative Kubernetes Usage

Imperative

Declarative

`kubectl create deployment ...`

Individual commands are executed to trigger certain Kubernetes actions

Comparable to using `docker run` only

`kubectl apply -f config.yaml`

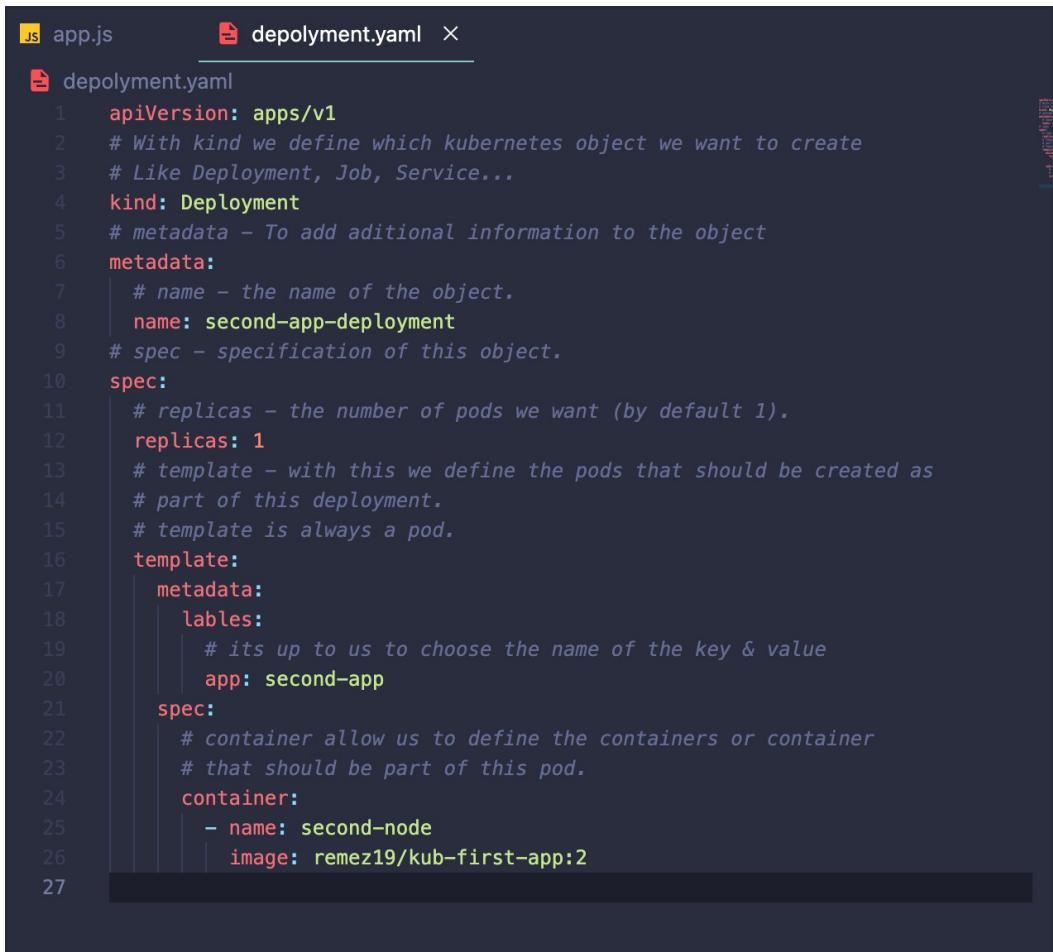
A config file is defined and applied to change the desired state

Comparable to using Docker Compose with compose files

Creating a Deployment Configuration File (Declarative Approach):

```
JS app.js          deployment.yaml ×
  deployment.yaml
  1  apiVersion: apps/v1
  2  # With kind we define which kubernetes object we want to create
  3  # Like Deployment, Job, Service...
  4  kind: Deployment
  5  # To add aditional information to the object we create we use metadata
  6  metadata:
  7    |   name: second-app-deployment
  8    # The specification of this object.
  9    spec:
10    |
```

Adding Pod and Container Specs:



The screenshot shows a terminal window with two tabs open: "app.js" and "deployment.yaml". The "deployment.yaml" tab is active and displays the following YAML configuration:

```
apiVersion: apps/v1
# With kind we define which kubernetes object we want to create
# Like Deployment, Job, Service...
kind: Deployment
# metadata - To add aditional information to the object
metadata:
  # name - the name of the object.
  name: second-app-deployment
# spec - specification of this object.
spec:
  # replicas - the number of pods we want (by default 1).
  replicas: 1
  # template - with this we define the pods that should be created as
  # part of this deployment.
  # template is always a pod.
  template:
    metadata:
      labels:
        # its up to us to choose the name of the key & value
        app: second-app
    spec:
      # container allow us to define the containers or container
      # that should be part of this pod.
      container:
        - name: second-node
          image: remez19/kub-first-app:2
```

Continue:

```

d.yaml  X
d.yaml
1  apiVersion: apps/v1
2  # With kind we define which kubernetes object we want to create
3  # Like Deployment, Job, Service...
4  kind: Deployment
5  # metadata - To add additional information to the object
6  metadata:
7    # name - the name of the object.
8    name: second-app-deployment
9  # spec - specification of this object.
10 spec:
11   # replicas - the number of pods we want (by default 1).
12   replicas: 1
13   # With the selector we specify which pods should be managed by this deployment.
14   selector:
15     matchLabels:
16       app: second-app
17   # template - with this we define the pods that should be created as
18   # part of this deployment.
19   # template is always a pod.
20   template:           Tells the deployment which pods belong to it.
21     metadata:
22       labels:
23         # its up to us to choose the name of the key & value
24         app: second-app
25   spec:
26     # container allow us to define the containers or container
27     # that should be part of this pod.
28     container:
29       - name: second-node
30         image: remez19/kub-first-app:2
31

```

To use the file - "kubectl apply -f=[PATH_TO_FILE]".

Example - "kubectl apply -f=d.yaml"

Creating a Service Declaratively:

```

service.yaml  X
service.yaml
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: backend
5  spec:
6    selector:
7      # Which pods should be managed by this service
8      app: second-app
9    ports:
10      - protocol: "TCP"
11        port: 80
12        targetPort: 8080
13        type: LoadBalancer
14

```

Using this file - "kubectl apply -f=service.yaml"

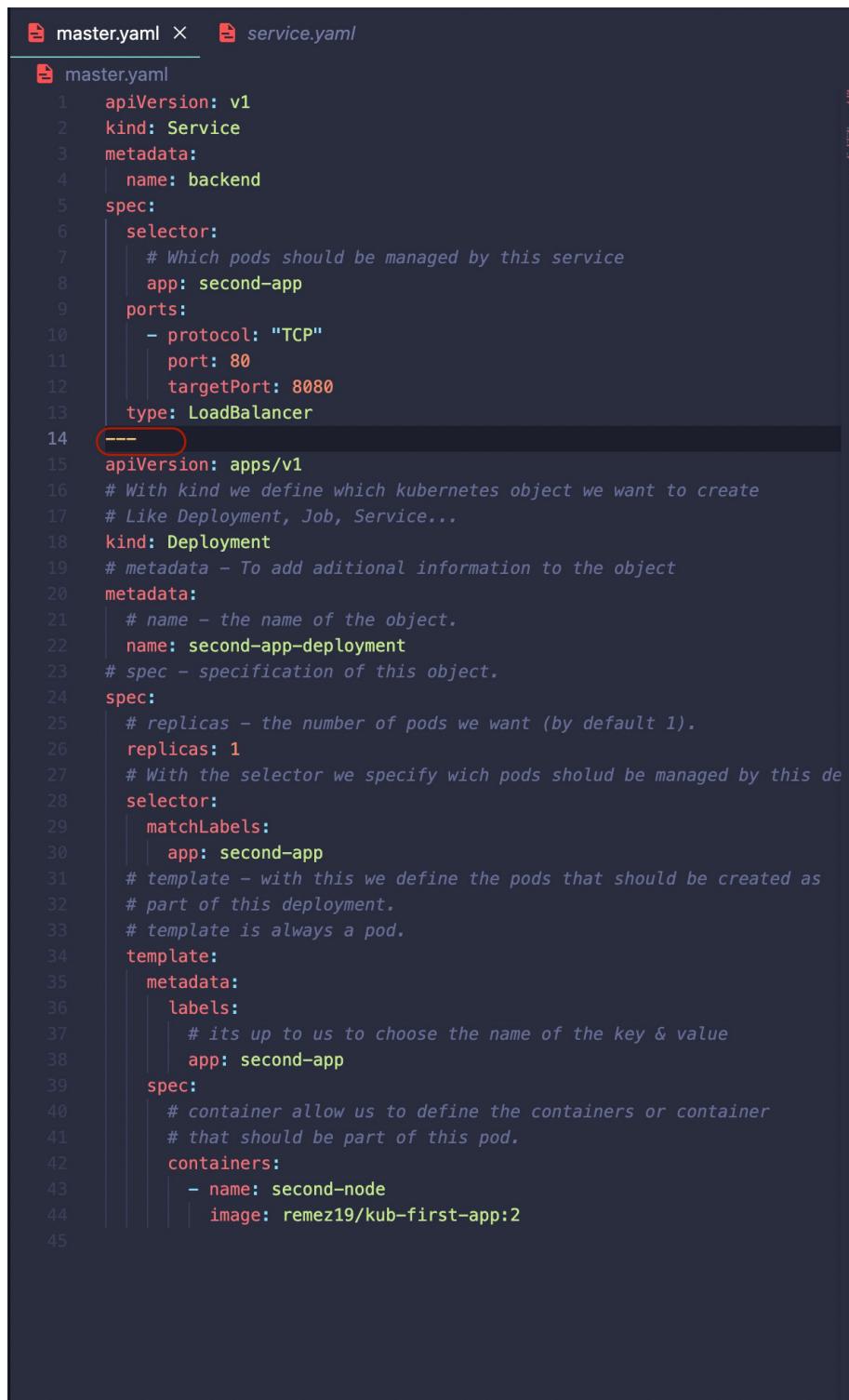
Updating & Deleting Resources:

To make changes we can change our file however we want (create any change we want) and, run "**kubectl apply -f=service.yaml**".

Multiple vs Single Config Files:

Merging the two file into one file.

Separating the objects with "—"

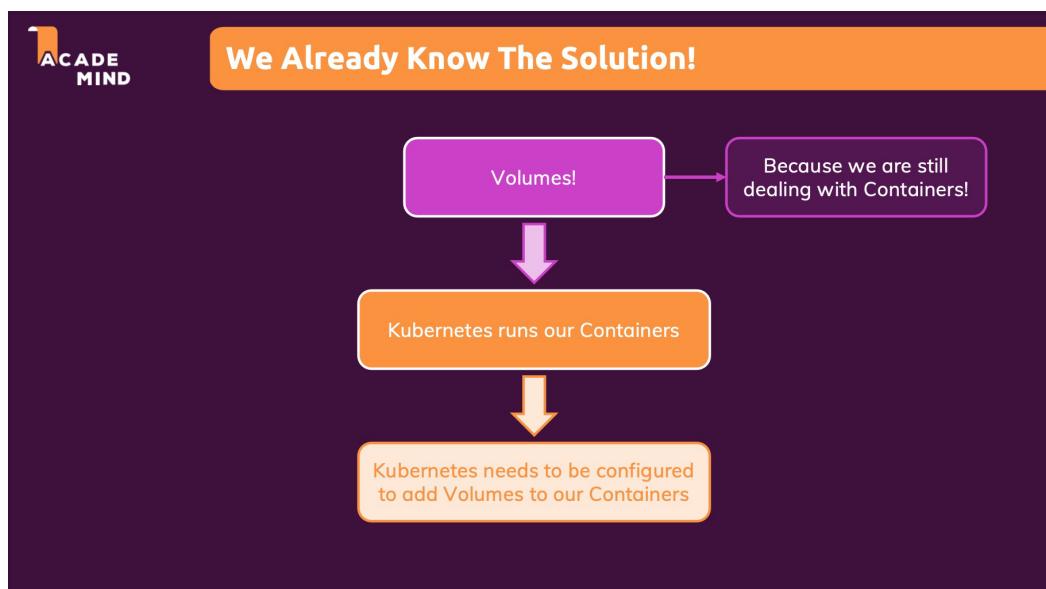
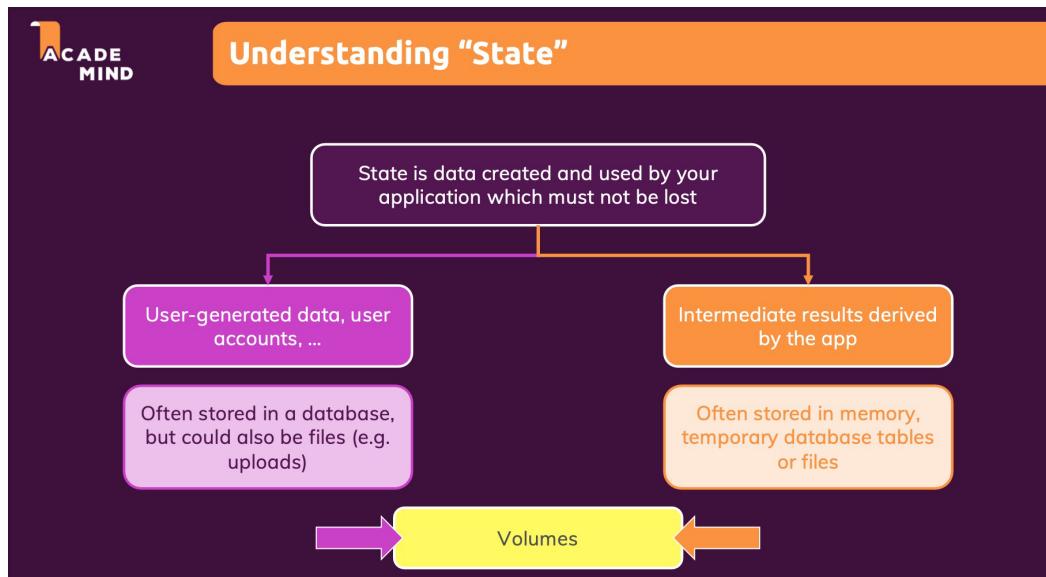


```
master.yaml *  service.yaml

master.yaml
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: backend
5  spec:
6    selector:
7      # Which pods should be managed by this service
8      app: second-app
9    ports:
10      - protocol: "TCP"
11        port: 80
12        targetPort: 8080
13      type: LoadBalancer
14  ---
15  apiVersion: apps/v1
16  # With kind we define which kubernetes object we want to create
17  # Like Deployment, Job, Service...
18  kind: Deployment
19  # metadata - To add additional information to the object
20  metadata:
21    # name - the name of the object.
22    name: second-app-deployment
23  # spec - specification of this object.
24  spec:
25    # replicas - the number of pods we want (by default 1).
26    replicas: 1
27    # With the selector we specify which pods should be managed by this deployment
28    selector:
29      matchLabels:
30        app: second-app
31    # template - with this we define the pods that should be created as
32    # part of this deployment.
33    # template is always a pod.
34    template:
35      metadata:
36        labels:
37          # its up to us to choose the name of the key & value
38          app: second-app
39    spec:
40      # container allow us to define the containers or container
41      # that should be part of this pod.
42      containers:
43        - name: second-node
44          image: remez19/kub-first-app:2
```

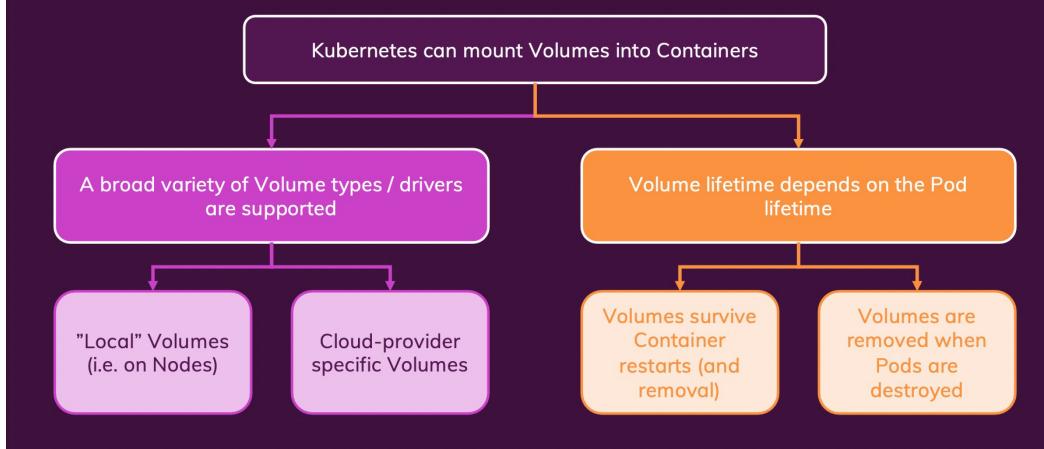
Section 13: Managing Data & Volumes with Kubernetes:

Kubernetes & Volumes - More Than Docker Volumes:



Kubernetes Volumes: Theory & Docker Comparison:

Kubernetes & Volumes



Kubernetes Volumes vs Docker Volumes

Kubernetes Volumes	Docker Volumes
Supports many different Drivers and Types	Basically no Driver / Type Support
Volumes are not necessarily persistent	Volumes persist until manually cleared
Volumes survive Container restarts and removals	Volumes survive Container restarts and removals

Creating a New Deployment & Service:

The screenshot shows a code editor with two tabs open: 'deploy.yaml' and 'service.yaml'. The 'deploy.yaml' file contains the following YAML configuration:

```
1 apiVersion: apps/v1
2 kind: Deployment
3 metadata:
4   name: story-deployment
5 spec:
6   replicas: 1
7   selector:
8     matchLabels:
9       app: story
10  template:
11    metadata:
12      labels:
13        app: story
14  spec:
15    containers:
16      - name: story
17        image: remez19/kub-volumes
```

The 'service.yaml' file contains the following YAML configuration:

```
1 apiVersion: v1
2 kind: Service
3 metadata:
4   name: story-service
5 spec:
6   selector:
7     app: story
8   type: LoadBalancer
9   ports:
10    - protocol: "TCP"
11      port: 80
12      targetPort: 3000
```

Getting Started with Kubernetes Volumes:

The official documentation - <https://kubernetes.io/docs/concepts/storage/volumes/>

First Volume: The "emptyDir" Type:

```
deploy.yaml ×
deploy.yaml
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: story-deployment
5  spec:
6    replicas: 1
7    selector:
8      matchLabels:
9        app: story
10   template:
11     metadata:
12       labels:
13         app: story
14     spec:
15       containers:
16         - name: story
17           image: remez19/kub-volumes:1
18           # Making the volume available to this container.
19           volumeMounts:
20             # Where the data should be saved.
21             - mountPath: /app/story
22               name: story-volume
23       volumes:
24         - name: story-volume
25           # emptyDir creates a new empty directory whenever the pod starts
26           # and it keeps this directory alive and fill with data as long
27           # as the pod alive.
28           # Containers can write/read to this directory and, if containers stops
29           # or removed the data survive but if the pod is removed the directory
30           # also removed. when the pod is recreated the directory is recreated.
31           emptyDir: {}
32
```

A Second Volume: The "hostPath" Type:

```
deploy.yaml ×
deploy.yaml
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: story-deployment
5  spec:
6    replicas: 1
7    selector:
8      matchLabels:
9        app: story
10   template:
11     metadata:
12       labels:
13         app: story
14     spec:
15       containers:
16         - name: story
17           image: remez19/kub-volumes:1
18           # Making the volume available to this container.
19           volumeMounts:
20             # Where the data should be saved.
21             - mountPath: /app/story
22               name: story-volume
23       volumes:
24         - name: story-volume
25           # emptyDir creates a new empty directory whenever the pod starts
26           # and it keeps this directory alive and fill with data as long
27           # as the pod alive.
28           # Containers can write/read to this directory and, if containers stops
29           # or removed the data survive but if the pod is removed the directory is
30           # also removed. when the pod is recreated the directory is recreated.
31           # emptyDir: {}
32           # hostPath - allows us to set a path on the host machine on the
33           # Node running this pod and then the data from this path will be exposed to
34           # the other pods.
35           hostPath:
36             # The path on the host machine where data should be stored.
37             # This works like a bind-mount.
38             # We can use this approach to also share data on our
39             # host machine with the containers.
40             path: /data
41             # With type we let kubernetes know how this path should be handled
42             # for example - if the directory should be created if it doesn't exist.
43             type: DirectoryOrCreate
44
```

Understanding the "CSI" Volume Type:

The CSI volume type is a very flexible type which allows us to attach any storage solution out there in the world as long as there is an integration for this SCI type.

From Volumes to Persistent Volumes:

Up until now all the solutions we saw (volume types) have one disadvantage - they are destroyed when a pod is removed.

The hostPath works fine because we are using Minikube which contains only one Node (we only have one worker node).

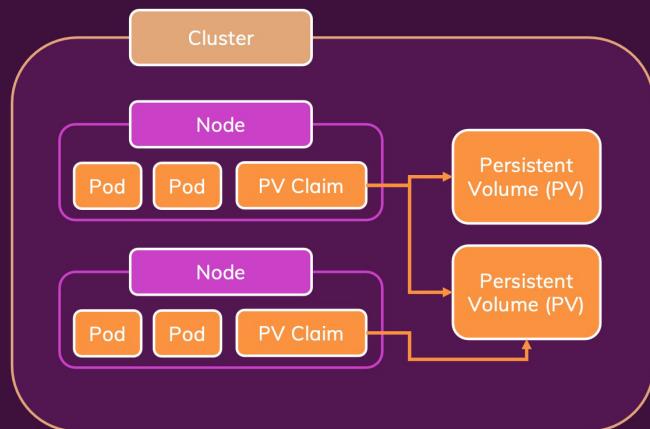
Persistent Volumes

Volumes are destroyed when a Pod is removed → hostPath partially works around that in "One-Node" environments

Pod- and Node-independent Volumes are sometimes required

Persistent Volumes

Persistent Volumes & Persistent Volume Claims



Defining a Persistent Volume:

```
host-pv.yaml ×
host-pv.yaml
1 # Setting a Defining a Persistent Volume
2 # With the hostPath type/driver volume.
3 apiVersion: v1
4 kind: PersistentVolume
5 metadata:
6   name: host-pv
7 spec:
8   # The capacity is important because we as admins want to controll how much capacity
9   # can be used by the diffrent pods.
10  capacity:
11    # The overall storage that can be used.
12    # 4Gi - 4 Giga 4Gi
13    storage: 1Gi
14    # volumeMode - two optnios Filesystem/Block.
15    # This are diffrent storage types.
16    volumeMode: Filesystem
17    # Define how this volume should be accessed.
18    # We have three available options:
19    # ReadWriteOnce - the volume can be accessed by only one node.
20    # ReadOnlyMany - Rean only by many Nodes (cant work with hostPath - we have only
21    # one Node).
22    # ReadWriteMany
23    accessMode:
24      - ReadWriteOnce
25    # This will setup a hosPath persistent volume as a stnd alone
26    # detached from any pod in this enviroment. (only work when we have a single Node)
27    hostPath:
28      path: /data
29      type: DirectoryOrCreate
30
```

Creating a Persistent Volume Claim:

This approach make sure that we have Persistent volume that is independent from the other nodes.

Persistent Volume

```

host-pvc.yaml deploy.yaml host-pv.yaml ×
host-pv.yaml

1 # Setting a Defining a Persistent Volume
2 # With the hostPath type/driver volume.
3 apiVersion: v1
4 kind: PersistentVolume
5 metadata:
6   name: host-pv
7 spec:
8   # The capacity is important because we as admins want to controll how much capacity
9   # can be used by the diffrent pods.
10  capacity:
11    # The overall storage that can be used.
12    # 4Gi - 4 Giga 4Gi
13    storage: 1Gi
14    # volumeMode - two optnios Filesystem/Block.
15    # This are diffrent storage types.
16    volumeMode: Filesystem
17    storageClassName: standard
18    # Define how this volume should be accessed.
19    # We have three available options:
20    # ReadWriteOnce - the volume can be accessed by only one node.
21    # ReadOnlyMany - Rean only by many Nodes (cant work with hostPath - we have only
22    # one Node).
23    # ReadWriteMany
24    accessMode:
25      - ReadWriteOnce
26    # This will setup a hosPath persistent volume as a stnd alone
27    # detached from any pod in this enviroment. (only work when we have a single Node)
28    hostPath:
29      path: /data
30      type: DirectoryOrCreate
31

```

Claiming the persistent volume

```

host-pvc.yaml × deploy.yaml host-pv.yaml
host-pvc.yaml

1 # The "Claim"
2 # 12:21
3
4 apiVersion: v1
5 kind: PersistentVolumeClaim
6 metadata:
7   name: host-pvc
8 spec:
9   # Here we need to tell which persistent volume we want to claim
10  # To connect the clain to the persistent volume we add the
11  # volumeName: [VOLUME_NAME]
12  volumeName: host-pv
13  # we define how we want to aceess this volume.
14  # we can have more then one way of accessing
15  accessMode:
16    - ReadWriteOnce
17  # resources - is the counterpart to "capacity".
18  storageClassName: standard
19  resources:
20    requests:
21      storage: 1Gi # the maximum available for us.
22

```

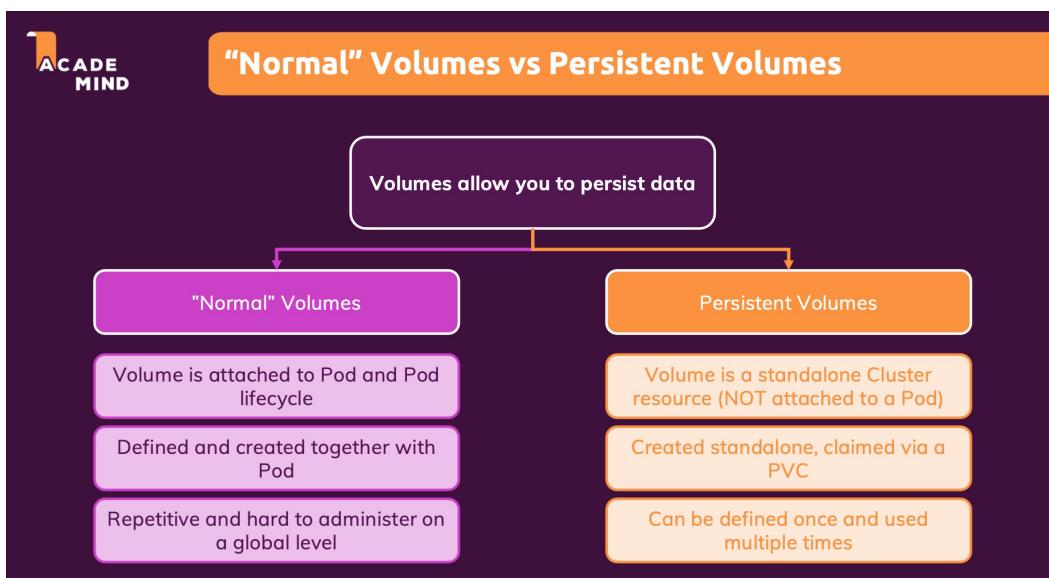
Connecting the pods to the claim we created.

```

host-pvc.yaml          deploy.yaml X      host-pv.yaml
deploy.yaml
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: story-deployment
5  spec:
6    replicas: 1
7    selector:
8      matchLabels:
9        app: story
10   template:
11     metadata:
12       labels:
13         app: story
14     spec:
15       containers:
16         - name: story
17           image: remez19/kub-volumes:1
18           # Making the volume available to this container.
19           volumeMounts:
20             # Where the data should be saved.
21             - mountPath: /app/story
22               name: story-volume
23
24   volumes:
25     - name: story-volume
26       # Connecting the pod to a claim (Persistent Volume)
27       persistentVolumeClaim:
28         claimName: host-pvc

```

Volumes vs Persistent Volumes:



Using Environment Variables:

```

deploy.yaml
...
  env:
    - name: STORY_FOLDER
      value: "story"
      # Making the volume available to the container
      volumeMounts:
        - mountPath: /app/story
          name: story-volume
          # Where the data should be saved
          persistentVolumeClaim:
            claimName: host-pvc
app.js
...
  const filePath = path.join(__dirname, process.env.STORY_FOLDER, "text.txt");
  app.use(bodyParser.json());
  app.get("/story", (req, res) => {
    fs.readFile(filePath, (err, data) => {
      if (err) {
        return res.status(500).json({ message: "Failed to open file." });
      }
      res.status(200).json({ story: data.toString() });
    });
  });
  app.post("/story", (req, res) => {
    const newText = req.body.text;
    if (newText.trim().length === 0) {
      return res.status(422).json({ message: "Text must not be empty!" });
    }
    fs.appendFile(filePath, newText + "\n", (err) => {
      if (err) {
        return res.status(500).json({ message: "Storing the text failed." });
      }
      res.status(201).json({ message: "Text was stored!" });
    });
  });
}

```

Environment Variables & ConfigMaps:

<https://www.udemy.com/course/docker-kubernetes-the-practical-guide/learn/lecture/22627837#overview>

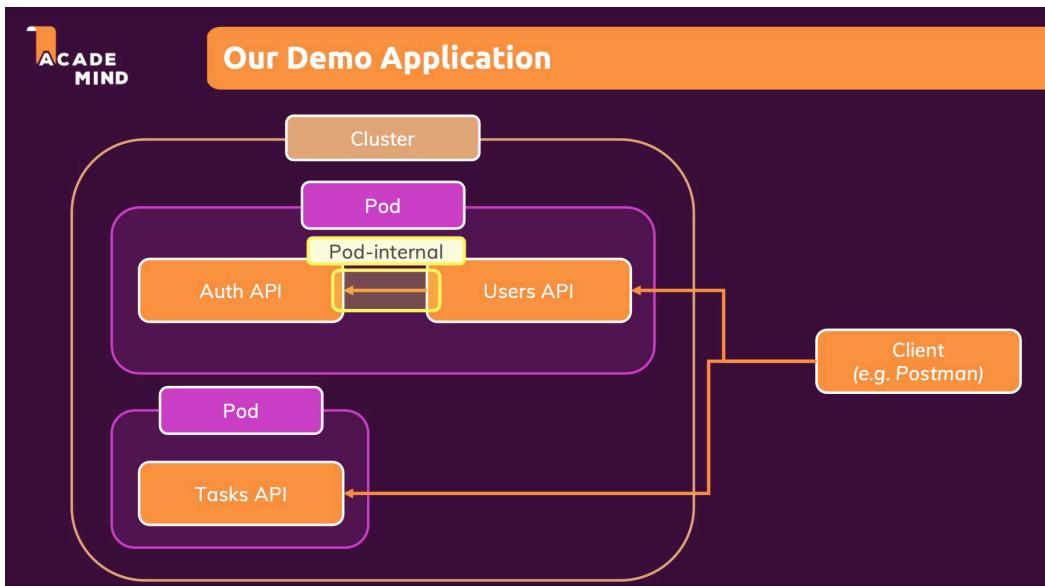
```

deploy.yaml
...
  env:
    - name: STORY_FOLDER
      valueFrom:
        configMapKeyRef:
          name: data-store-env
          key: folder
          # Making the volume available to the container
          volumeMounts:
            - mountPath: /app/story
              name: story-volume
              # Where the data should be saved
              persistentVolumeClaim:
                claimName: host-pvc
environment.yaml
...
apiVersion: v1
kind: ConfigMap
metadata:
  name: data-store-env
data:
  folder: "story"

```

Section 14: Kubernetes Networking:

Starting Project & Our Goal:



Creating a First Deployment:

```

EXPLORER      ...
OPEN EDITORS
  users-deployment.yaml
KUB-NETWORK-01-STA...
  auth-api
  kubernetes
    users-deployment.yaml
  tasks-api
  users-api
  docker-compose.yml

  users-deployment.yaml ×
kubernetes > users-deployment.yaml
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: users-deployment
5  spec:
6    replicas: 1
7    selector:
8      matchLabels:
9        app: users
10   template:
11     metadata:
12       labels:
13         app: users
14     spec:
15       containers:
16         - name: users
17           image: remez19/kub-demo-users
18

```

Another Look at Services:

```
EXPLORER      ...
OPEN EDITORS   ...
KUB-NETWORK-01-STA...
  auth-api
  kubernetes
    users-deployment.yaml
    users-service.yaml
  tasks-api
  users-api
  docker-compose.yml

users-service.yaml
kubernetes > users-service.yaml
1  apiVersion: v1
2  kind: Service
3  metadata:
4  | name: users-service
5  spec:
6  | selector:
7  | | app: users
8  | type: LoadBalancer
9  ports:
10 | - protocol: TCP
11 | | port: 8080
12 | # targetPort - the container internal port in which it expect incoming
13 | # requests.
14 | | targetPort: 8080
```

Multiple Containers in One Pod:

```
users-deployment.yaml
kubernetes > users-deployment.yaml
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4  | name: users-deployment
5  spec:
6  | replicas: 1
7  | selector:
8  | | matchLabels:
9  | | | app: users
10 | template:
11 | | metadata:
12 | | | labels:
13 | | | | app: users
14 | | spec:
15 | | | containers:
16 | | | | - name: users
17 | | | | | image: remez19/kub-demo-users:latest
18 | | | | - name: auth
19 | | | | | image: remez19/kub-demo-auth:latest
```

Pod-internal Communication:

Two containers in one pod communication.

Since we need one value in production and one value in development we use environment variables.

```
users-deployment.yaml ×  
kubernetes > users-deployment.yaml  
1  apiVersion: apps/v1  
2  kind: Deployment  
3  metadata:  
4    name: users-deployment  
5  spec:  
6    replicas: 1  
7    selector:  
8      matchLabels:  
9        app: users  
10   template:  
11     metadata:  
12       labels:  
13         app: users  
14     spec:  
15       containers:  
16         - name: users  
17           image: remez19/kub-demo-users:latest  
18           env:  
19             - name: AUTH_ADDRESS  
20               value: localhost  
21             - name: auth  
22               image: remez19/kub-demo-auth:latest  
23
```

```

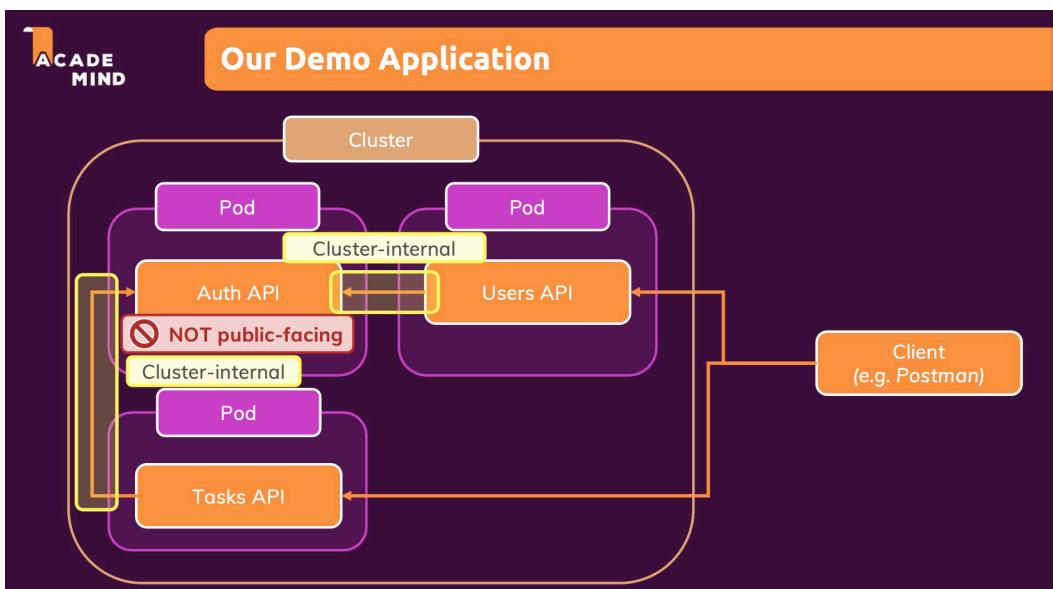
users-app.js ×

users-api > users-app.js > app.post("/login") callback > response
1 const express = require("express");
2 const bodyParser = require("body-parser");
3 const axios = require("axios");
4
5 const app = express();
6
7 app.use(bodyParser.json());
8
9 app.post("/signup", async (req, res) => {
10   // It's just a dummy service - we don't really care for the email
11   const email = req.body.email;
12   const password = req.body.password;
13
14   if (
15     !password ||
16     password.trim().length === 0 ||
17     !email ||
18     email.trim().length === 0
19   ) {
20     return res
21       .status(422)
22       .json({ message: "An email and password needs to be specified!" });
23   }
24
25   try {
26     const hashedPW = await axios.get(
27       `http://${process.env.AUTH_ADDRESS}/hashed-password/` + password
28     );
29     // const hashedPW = "dummy text";
30     // since it's a dummy service, we don't really care for the hashed-pw either
31     console.log(hashedPW, email);
32     res.status(201).json({ message: "User created!" });
33   }
34 }

```

Creating Multiple Deployments:

We are going to look at pod-to-pod communication.



Pod-to-Pod Communication with IP Addresses & Environment Variables:

232:

<https://www.udemy.com/course/docker-kubernetes-the-practical-guide/learn/>

[lecture/22627933#overview](https://www.udemy.com/course/docker-kubernetes-the-practical-guide/learn/lecture/22627933#overview)

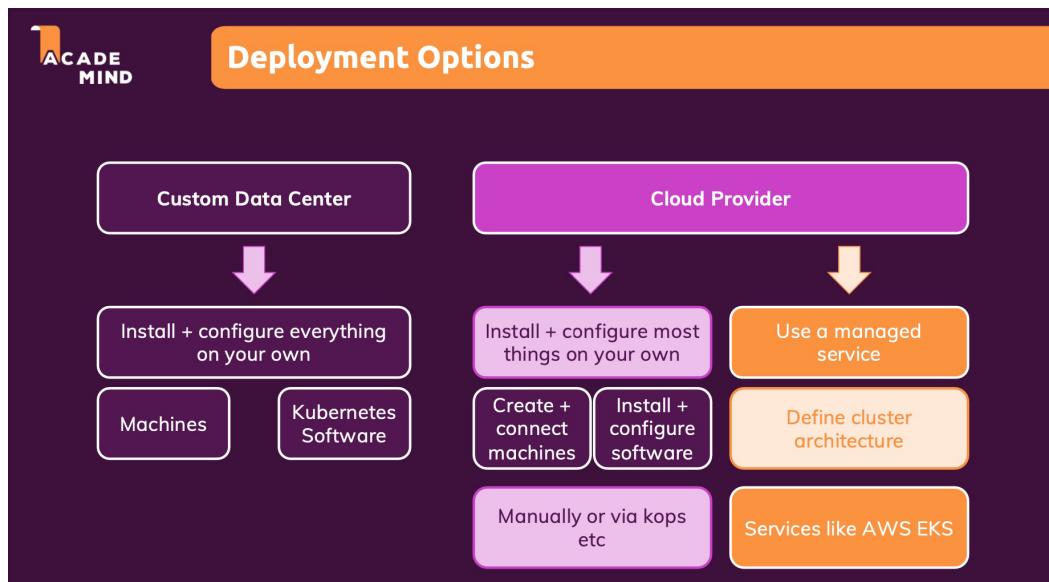
Using DNS for Pod-to-Pod Communication:

233

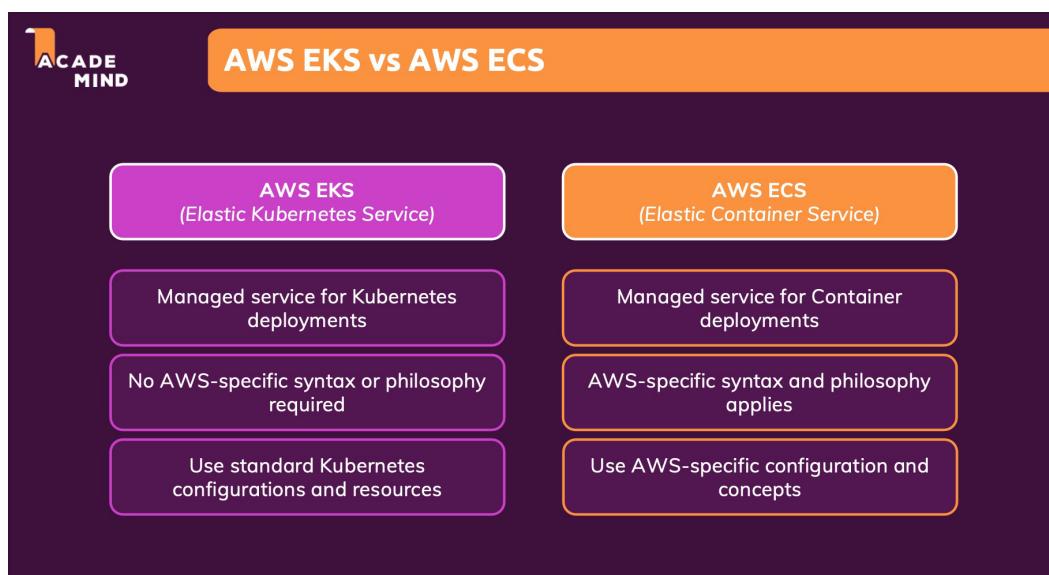
<https://www.udemy.com/course/docker-kubernetes-the-practical-guide/learn/lecture/22627937#overview>

Section 15: Kubernetes - Deployment (AWS EKS):

Deployment Options & Steps:



AWS EKS vs AWS ECS:



Creating & Configuring the Kubernetes Cluster with EKS:

Cluster service role - in order to allow EKS to create other resources on our behalf we need to give the appropriate permissions.

We do that with another AWS service which allows us to manage

permissions - IAM (Identity Access Management) service.

EKS > Clusters > Create EKS cluster

Step 1
Configure cluster

Step 2
Specify networking

Step 3
Configure logging

Step 4
Review and create

Configure cluster

Cluster configuration Info

Name
Enter a unique name for this cluster. This property cannot be changed after the cluster is created.

The cluster name should begin with letter or digit and can have any of the following characters: the set of Unicode letters, digits, hyphens and underscores. Maximum length of 100.

Kubernetes version Info
Select the Kubernetes version for this cluster.

Cluster service role Info
Select the IAM role to allow the Kubernetes control plane to manage AWS resources on your behalf. This property cannot be changed after the cluster is created. To create a new role, follow the instructions in the [Amazon EKS User Guide](#).

Secrets encryption Info
Once turned on, secrets encryption cannot be modified or removed.

Turn on envelope encryption of Kubernetes secrets using KMS
Envelope encryption provides an additional layer of encryption for your Kubernetes secrets.

Tags (0) Info

This cluster does not have any tags.

Remaining tags available to add: 50

IAM Management Console Cluster Create Docker Hub Database Deployments | CloudWatch Metrics for services, features, blogs, docs, and more [Option+S] Global Remez1

New! Securely access AWS services from your data center with IAM Roles Anywhere. [Learn more](#)

IAM > Roles

Roles (2) Info

An IAM role is an identity you can create that has specific permissions with credentials that are valid for short durations. Roles can be assumed by entities that you trust.

Role name	Trusted entities	Last activity
AWSServiceRoleForSupport	AWS Service: support (Service-Linked Role)	-

EKS Cluster - this will give us a pre-defined role which is pre-defined for EKS to give EKS all the permissions it needs

Screenshot of the AWS IAM Management Console "Select trusted entity" page.

The top navigation bar includes links for IAM Management Console, Cluster Create, Docker Hub, Database Deployments, Global, and a user named Remez1.

Select trusted entity Info

Trusted entity type

AWS service
Allow AWS services like EC2, Lambda, or others to perform actions in this account.

AWS account
Allow entities in other AWS accounts belonging to you or a 3rd party to perform actions in this account.

Web identity
Allows users federated by the specified external web identity provider to assume this role to perform actions in this account.

SAML 2.0 federation
Allow users federated with SAML 2.0 from a corporate directory to perform actions in this account.

Custom trust policy
Create a custom trust policy to enable others to perform actions in this account.

Use case

Allow an AWS service like EC2, Lambda, or others to perform actions in this account.

Common use cases

EC2
Allows EC2 instances to call AWS services on your behalf.

Lambda
Allows Lambda functions to call AWS services on your behalf.

Use cases for other AWS services:

EKS ▼

EKS
Allows EKS to manage clusters on your behalf.

EKS - Cluster
Allows access to other AWS service resources that are required to operate clusters managed by EKS.

EKS - Nodegroup
Allow EKS to manage nodegroups on your behalf.

EKS - Fargate pod
Allows access to other AWS service resources that are required to run Amazon EKS pods on AWS Fargate.