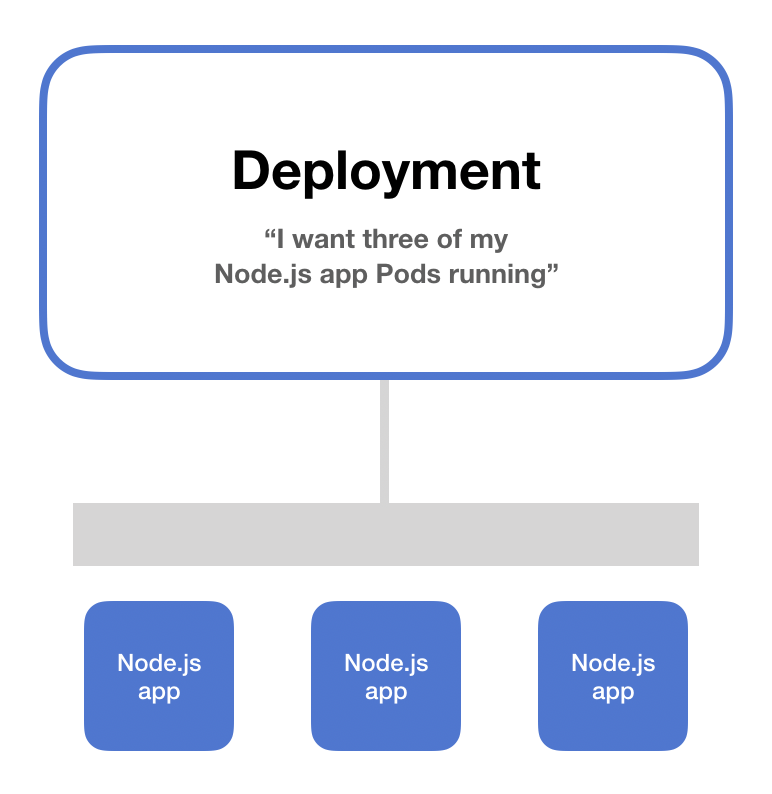
**Kubernetes Deployment with Example YAML**

Everyone running applications on Kubernetes cluster uses a deployment.

It’s what we use to scale, roll out, and roll back versions of your applications.

With a deployment, we tell Kubernetes how many copies of a Pod we want running. The deployment takes care of everything else.



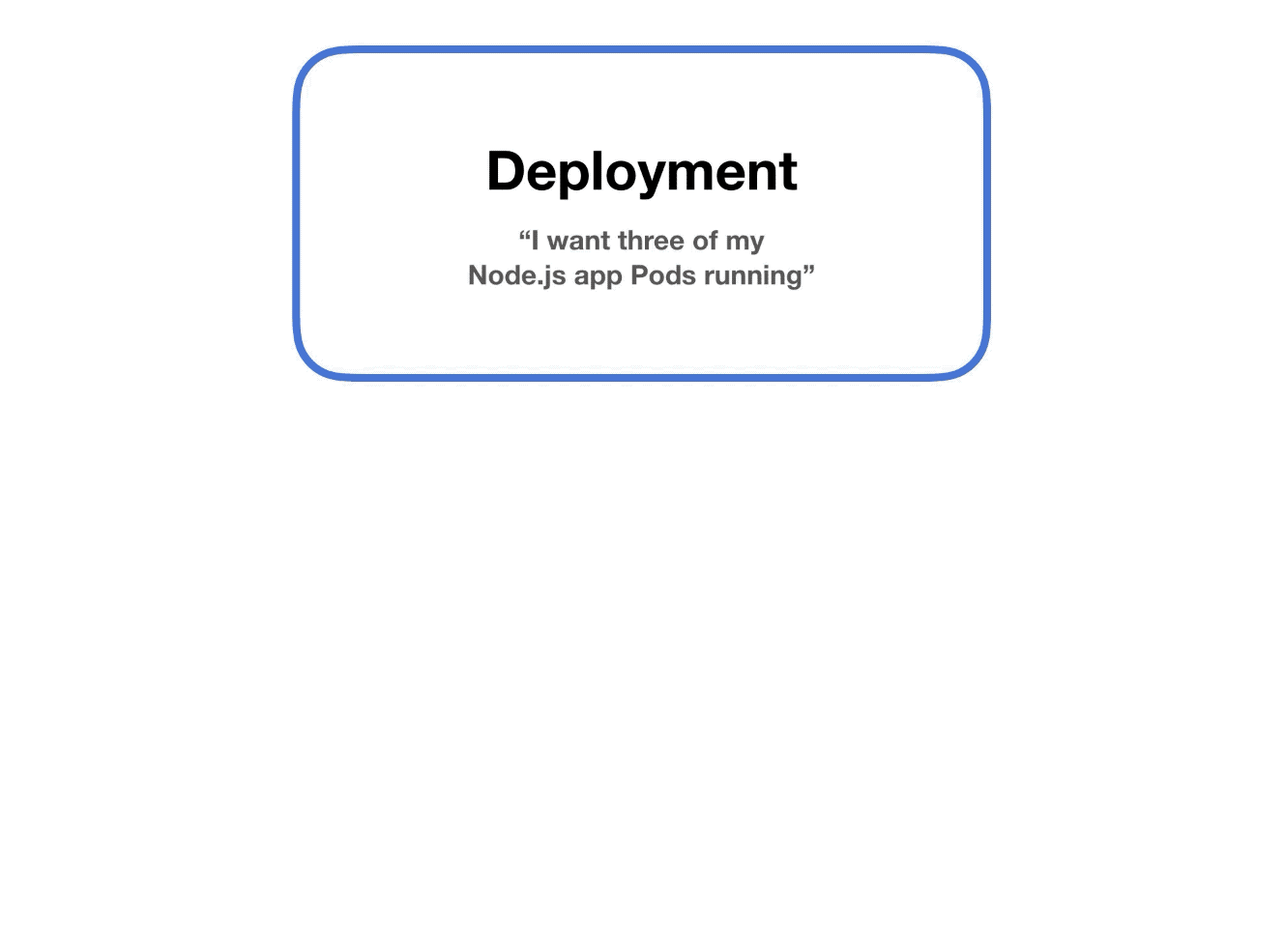
**What is a Deployment?**

A deployment is an object in Kubernetes that lets you manage a set of identical pods.

Without a deployment, we need to create, update, and delete a bunch of pods manually.

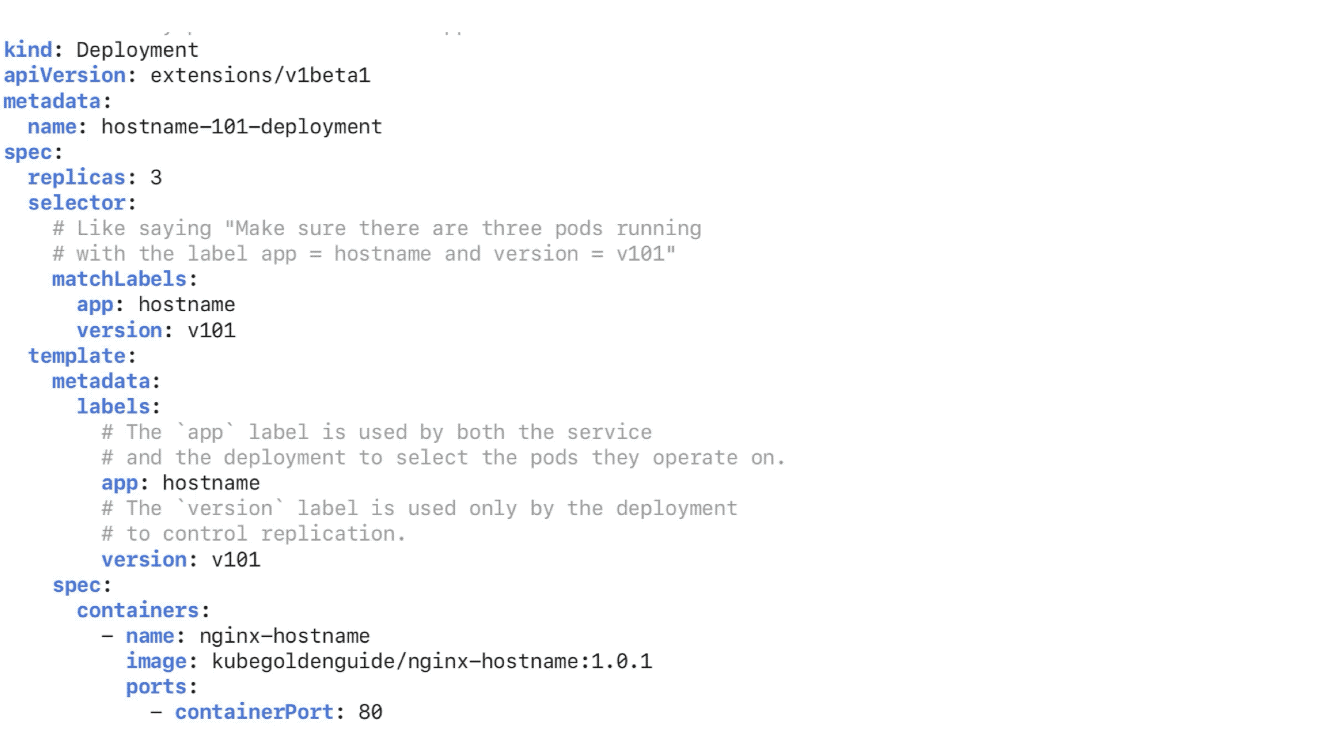
With a deployment, we declare a single object in a YAML file. This object is responsible for creating the pods, making sure they stay up to date, and ensuring there are enough of them running

You can also easily auto scale your applications using a Kubernetes deployment.



## YAML example

First, take a look at the animation that annotates each section of the deployment YAML. (Scroll down for code that can be copy-and-pasted.)



(You can ignore the additional comments about the "service" here – this deployment was taken from a different example that also incorporated services.)

First, the replicas key sets the number of instances of the pod that the deployment should keep alive.

Next, we use a label selector to tell the deployment which pods are part of the deployment. This essentially says "all the pods matching these labels are grouped in our deployment."

After that, we have the template object.

This is interesting. It’s essentially a pod template jammed inside our deployment spec. When the deployment creates pods, it will create them using this template!

So everything under the template key is a regular pod specification.

In this case, the deployment will create pods that run nginx-hostname and with the configured labels.

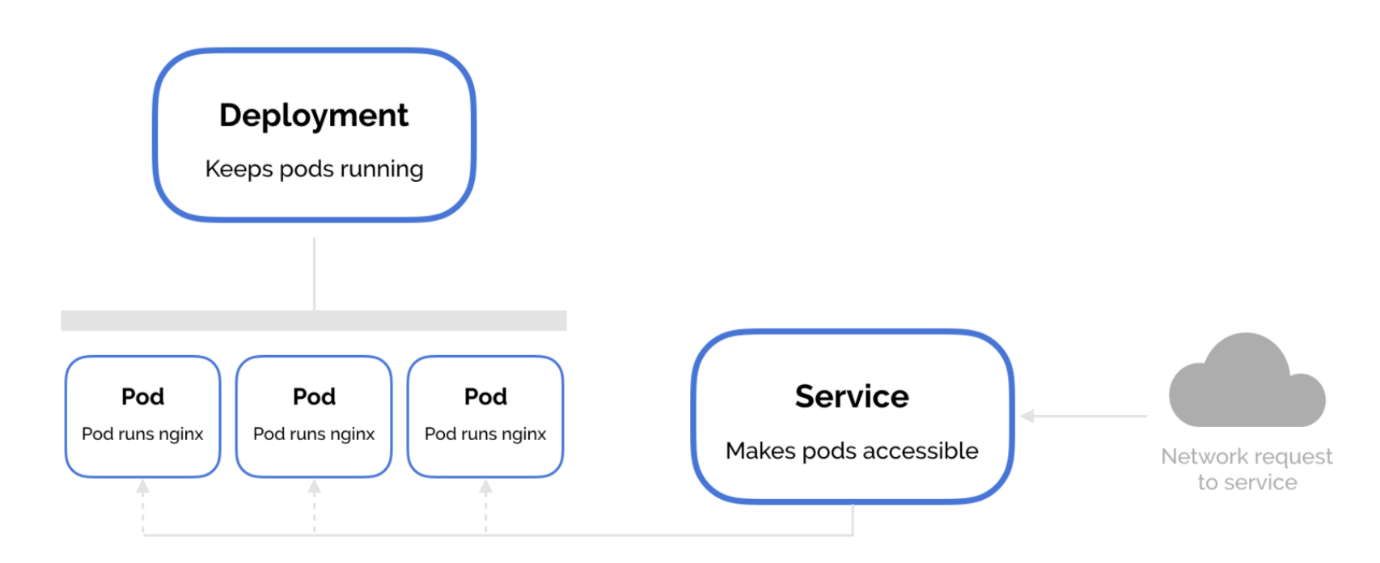
|  |  |
| --- | --- |
|  | kind: Deployment |
|  | apiVersion: extensions/v1beta1 |
|  | metadata: |
|  | name: nginx-deployment |
|  | spec: |
|  | # A deployment's specification really only |
|  | # has a few useful options |
|  |  |
|  | # 1. How many copies of each pod do we want? |
|  | replicas: 3 |
|  |  |
|  | # 2. How do want to update the pods? |
|  | strategy: Recreate |
|  |  |
|  | # 3. Which pods are managed by this deployment? |
|  | selector: |
|  | # This must match the labels we set on the pod! |
|  | matchLabels: |
|  | deploy: example |
|  |  |
|  | # This template field is a regular pod configuration |
|  | # nested inside the deployment spec |
|  | template: |
|  | metadata: |
|  | # Set labels on the pod. |
|  | # This is used in the deployment selector. |
|  | labels: |
|  | deploy: example |
|  | spec: |
|  | containers: |
|  | - name: nginx |
|  | image: nginx:1.7.9 |

## Deployment vs service

What’s the difference between a deployment and a service?

A deployment is used to keep a set of pods running by creating pods from a template.

A service is used to allow network access to a set of pods.



Both services and deployments choose which pods they operate on using labels and label selectors. This is where the overlap is.

You can use deployments independent of services. You can use services independent of deployments. They just go together really nicely!

This lets you do canary deployments. You add a new deployment version of a pod and run it alongside your existing deployment, but have both deployments handle requests to the service. Once deployed, you can verify the new deployment on a small proportion of the service's traffic, and gradually scale up the new deployment while decreasing the old one.

## How to delete a deployment in Kubernetes

1. If you’ve created your deployment from a file, you can use kubectl delete -f deployment.yaml to delete your deployment
2. If you’ve created your deployment from the command line, you can use kubectl delete deployment my-deployment to delete youMeet the Author