*Extending Kubernetes :*

1. CREATING A CUSTOMRESOURCEDEFINITION OBJECT :

* See website-crd.yaml in getRepo.

1. CREATING AN INSTANCE OF A CUSTOM RESOURCE :

* See kubia-website.yaml in getRepo.

1. Automating custom resources with custom controllers :

To make your Website objects run a web server pod exposed through a Service, you’ll need to build and deploy a Website controller, which will watch the API server for the

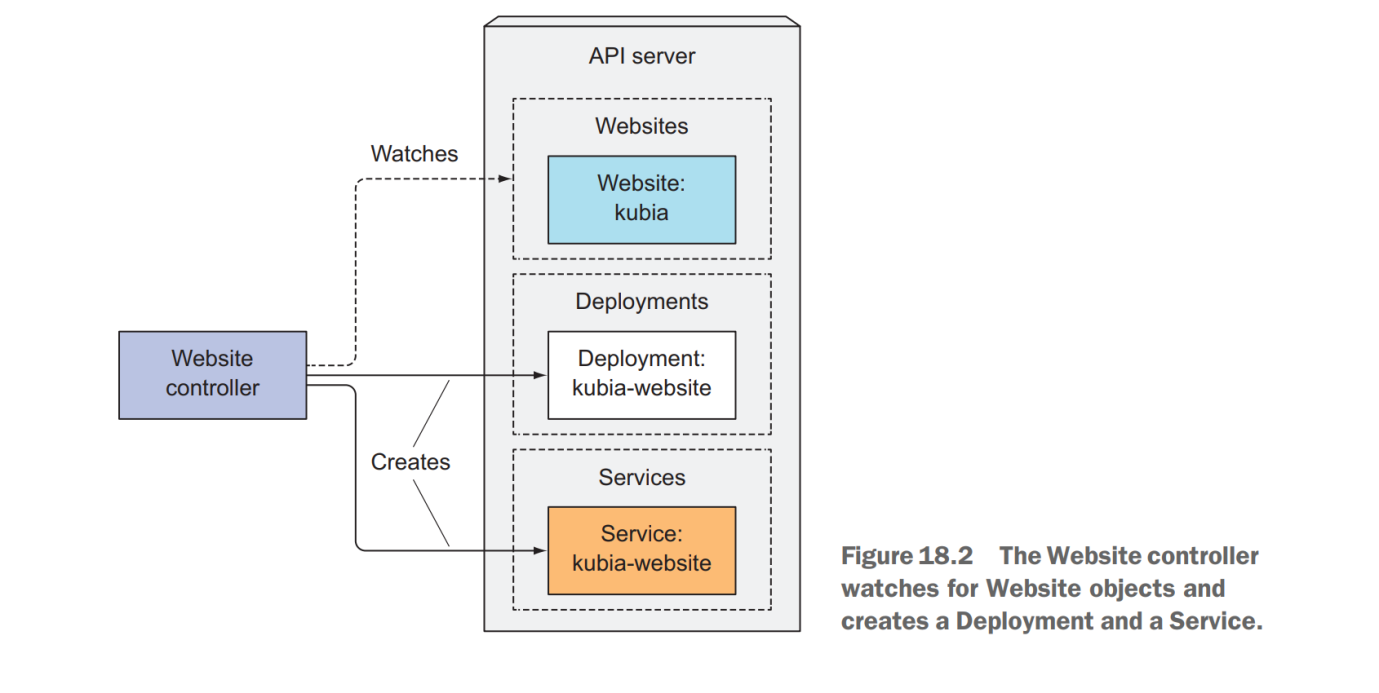
creation of Website objects and then create the Service and the web server Pod for

each of them.

To make sure the Pod is managed and survives node failures, the controller will

create a Deployment resource instead of an unmanaged Pod directly. The controller’s

operation is summarized in figure 18.2.



I’ve written a simple initial version of the controller, which works well enough to show CRDs and the controller in action, but it’s far from being production-ready, because it’s overly simplified. The container image is available at docker.io/luksa/ website-controller:latest, and the source code is at https://github.com/luksa/k8swebsite-controller. Instead of going through its source code, I’ll explain what the controller does.

UNDERSTANDING WHAT THE WEBSITE CONTROLLER DOES

Immediately upon startup, the controller starts to watch Website objects by requesting

the following URL:

http://localhost:8001/apis/extensions.example.com/v1/websites?watch=true

You may recognize the hostname and port—the controller isn’t connecting to the

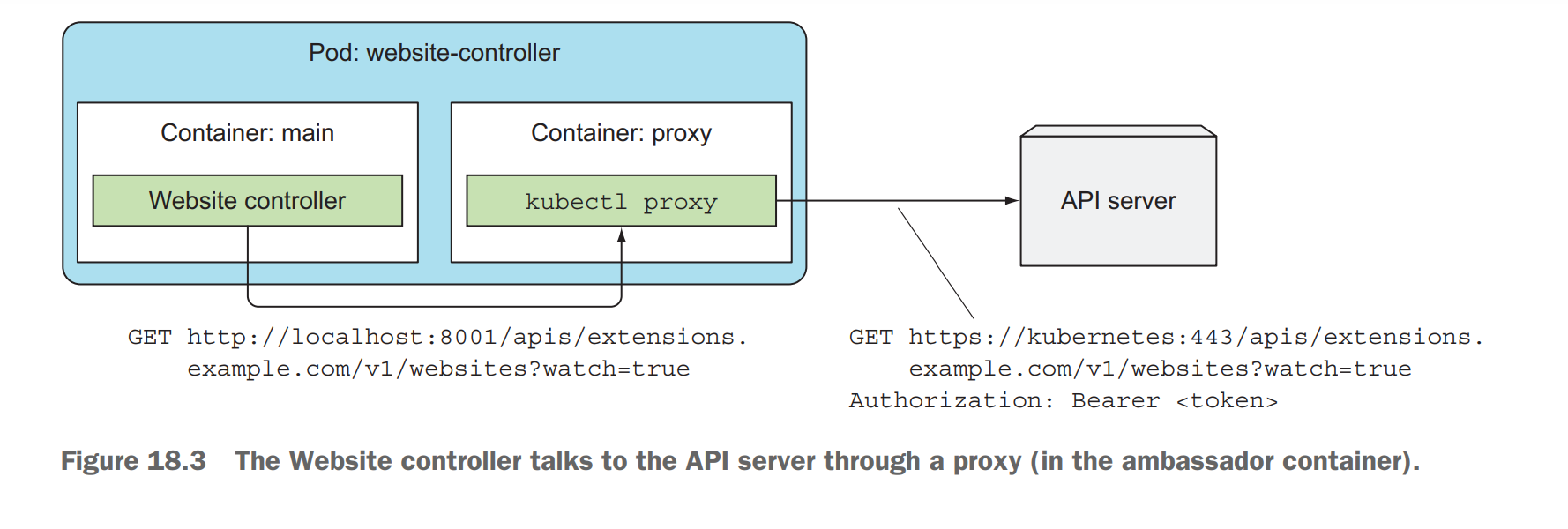
API server directly, but is instead connecting to the kubectl proxy process, which

runs in a sidecar container in the same pod and acts as the ambassador to the API

server (we examined the ambassador pattern in chapter 8). The proxy forwards the

request to the API server, taking care of both TLS encryption and authentication

(see figure 18.3).



Through the connection opened by this HTTP GET request, the API server will send

watch events for every change to any Website object.

The API server sends the ADDED watch event every time a new Website object is created. When the controller receives such an event, it extracts the Website’s name and

the URL of the Git repository from the Website object it received in the watch event

and creates a Deployment and a Service object by posting their JSON manifests to the

API server.

The Deployment resource contains a template for a pod with two containers

(shown in figure 18.4): one running an nginx server and another one running a gitsync process, which keeps a local directory synced with the contents of a Git repo.

The local directory is shared with the nginx container through an emptyDir volume

(you did something similar to that in chapter 6, but instead of keeping the local

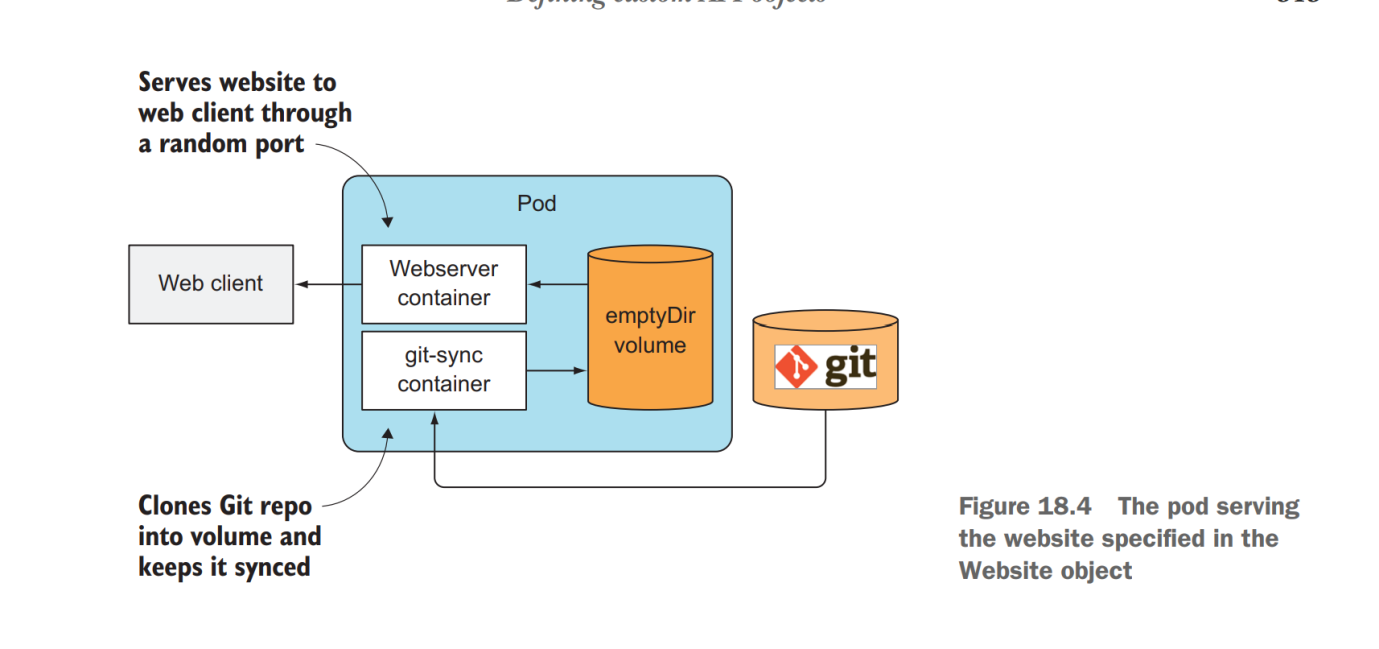
directory synced with a Git repo, you used a gitRepo volume to download the Git

repo’s contents at pod startup; the volume’s contents weren’t kept in sync with the

Git repo afterward). The Service is a NodePort Service, which exposes your web

server pod through a random port on each node (the same port is used on all

nodes). When a pod is created by the Deployment object, clients can access the website through the node port.



The next -> see the deployment of website-controller.yaml & logs after creating or deleting website custom resource