# <https://matthewpalmer.net/kubernetes-app-developer/articles/ultimate-configmap-guide-kubernetes.html>

**1. Define the ConfigMap in a YAML file.**

Create a YAML file setting the key-value pairs for your ConfigMap.

|  |  |
| --- | --- |
|  | kind: ConfigMap |
|  | apiVersion: v1 |
|  | metadata: |
|  | name: example-configmap |
|  | data: |
|  | # Configuration values can be set as key-value properties |
|  | database: mongodb |
|  | database\_uri: mongodb://localhost:27017 |
|  |  |
|  | # Or set as complete file contents (even JSON!) |
|  | keys: | |
|  | image.public.key=771 |
|  | rsa.public.key=42 |

**2. Create the ConfigMap in your Kubernetes cluster**

Create the ConfigMap using the command kubectl apply -f config-map.yaml

**3. Mount the ConfigMap through a Volume**

Each property name in this ConfigMap becomes a new file in the mounted directory (`/etc/config`) after you mount it.

|  |  |
| --- | --- |
|  | kind: Pod |
|  | apiVersion: v1 |
|  | metadata: |
|  | name: pod-using-configmap |
|  |  |
|  | spec: |
|  | # Add the ConfigMap as a volume to the Pod |
|  | volumes: |
|  | # `name` here must match the name |
|  | # specified in the volume mount |
|  | - name: example-configmap-volume |
|  | # Populate the volume with config map data |
|  | configMap: |
|  | # `name` here must match the name |
|  | # specified in the ConfigMap's YAML |
|  | name: example-configmap |
|  |  |
|  | containers: |
|  | - name: container-configmap |
|  | image: nginx:1.7.9 |
|  | # Mount the volume that contains the configuration data |
|  | # into your container filesystem |
|  | volumeMounts: |
|  | # `name` here must match the name |
|  | # from the volumes section of this pod |
|  | - name: example-configmap-volume |
|  | mountPath: /etc/config |

Attach to the created Pod using `kubectl exec -it pod-using-configmap sh`. Then run `ls /etc/config` and you can see each key from the ConfigMap added as a file in the directory. Use `cat` to look at the contents of each file and you’ll see the values from the ConfigMap.

You can then read the configuration settings using Python/Node.js/PHP from this file.

**How to use a ConfigMap with Environment Variables and `envFrom`?**

You can consume a ConfigMap via environment variables in a running container using the `envFrom` property.

**1. Create the ConfigMap**

Create the ConfigMap using the example from the previous section.

**2. Add the `envFrom` property to your Pod's YAML**

Set the `envFrom` key in each container to an object containing the list of ConfigMaps you want to include.

|  |  |
| --- | --- |
|  | kind: Pod |
|  | apiVersion: v1 |
|  | metadata: |
|  | name: pod-env-var |
|  | spec: |
|  | containers: |
|  | - name: env-var-configmap |
|  | image: nginx:1.7.9 |
|  | envFrom: |
|  | - configMapRef: |
|  | name: example-configmap |

[**view raw**](https://gist.github.com/matthewpalmer/c9649e7769e3e7e5e3790f5b3a4e6908/raw/c992552c067bd2a44d36c8c372381099d2678d77/pod-env-var.yaml)[**pod-env-var.yaml**](https://gist.github.com/matthewpalmer/c9649e7769e3e7e5e3790f5b3a4e6908#file-pod-env-var-yaml) hosted with ❤ by [**GitHub**](https://github.com/)

Attach to the created Pod using `kubectl exec -it pod-env-var sh`. Then run `env` and see that each key from the ConfigMap is now available as an environment variable.

**What are the other ways to create and use ConfigMaps?**

There are three other ways to create ConfigMaps using the `kubectl create configmap` command. I prefer the methods used above, but here are your options.

Use the contents of an entire directory with kubectl create configmap my-config --from-file=./my/dir/path/

Use the contents of a file or specific set of files with kubectl create configmap my-config --from-file=./my/file.txt

Use literal key-value pairs defined on the command line with kubectl create configmap my-config --from-literal=key1=value1 --from-literal=key2=value2

You can get more information about this command using kubectl create configmap --help.