# KUB 17: Ресурсы для хранения конфигурации

## Описание:

### ConfigMaps

Этот объект дает нам отделить конфигурацию от приложения. Мы можем записать k=v или файлы и затем отдать это приложению через kubernetes api.
Например, сделаем configmap из командной строки:

```
kubectl create configmap my-config --from-literal=key1=value1 --from-literal=key2=value2
```

И изучит, что получилось.

```
kubectl get configmaps my-config -o yaml
```

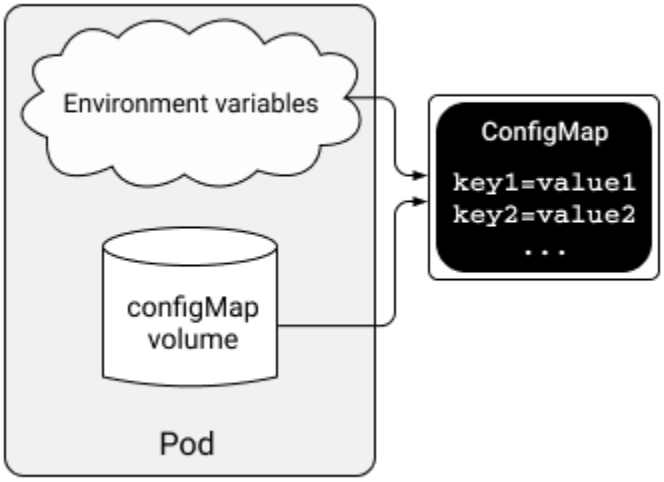
Обратите внимание на поле data. Можно создать любые Key=Value. Также можно создать configmap из файла.

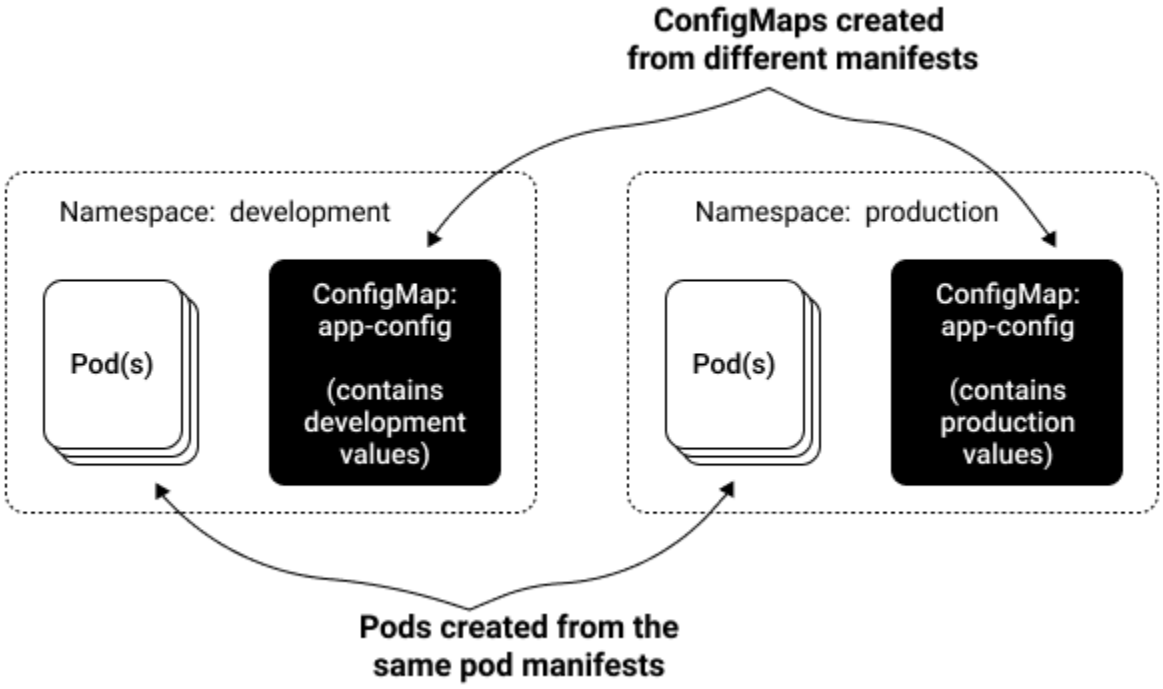
```
kubectl create configmap game-config --from-file=myconfig
```

Затем этот configmap можно использовать как env для контейнера:

```
containers:
  - name: rsvp-app
    image: teamcloudyuga/rsvpapp
    env:
    - name: MONGODB_HOST
      value: mongodb
    - name: TEXT1
      valueFrom:
        configMapKeyRef:
          name: customer1
          key: TEXT1
    - name: TEXT2
      valueFrom:
        configMapKeyRef:
          name: customer1
          key: TEXT2
    - name: COMPANY
```

```
    valueFrom:
      configMapKeyRef:
        name: customer1
        key: COMPANY
```




Или как Volume:

```
apiVersion: v1
kind: Pod
metadata:
```

```
  name: dapi-test-pod
spec:
  containers:
    - name: test-container
      image: k8s.gcr.io/busybox
      command: [ "/bin/sh", "-c", "ls /etc/config/" ]
      volumeMounts:
      - name: config-volume
        mountPath: /etc/config
  volumes:
    - name: config-volume
      configMap:
        name: special-config
  restartPolicy: Never
```

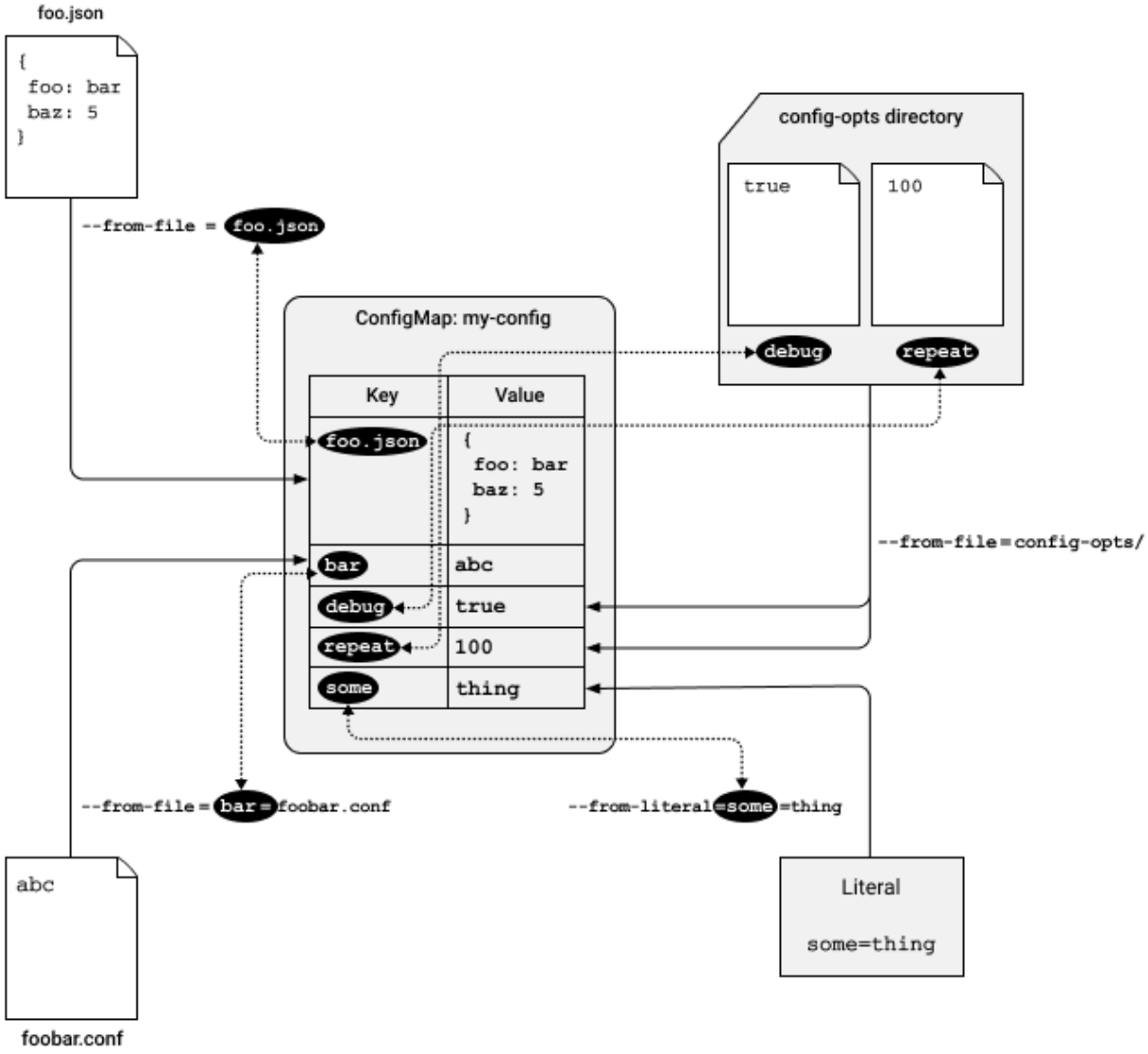
Во втором случае ключи карты будут видны как файлы по указанному пути. Если карта создавалась из файлов, они будут видны как обычные файлы.
В ходе монтирования Volume как директории, содержимое этой директории перезатирается, поэтому, если вам требуется сохранить его, но примонтировать конкретный файл из того же ConfigMap, то можно использовать в volumeMount директиву subPath.
Пример использования:

```
apiVersion: v1
kind: Pod
metadata:
  name: example
spec:
    containers:
    - name: test
      image: nginx:stable
      volumeMounts:
      - mountPath: /etc/nginx/sites-enabled/config.conf
        name: site-config
        subPath: config.conf
    volumes:
    - name: site-config
      configMap:
        name: test-config
```

ConfigMap может содержать как литералы, так и файлы, например:

```
kubectl create configmap my-config \
    --from-file=foo.json \
```

```
    --from-file=bar=foobar.conf \
    --from-file=config-opts/ \
    --from-literal=some=thing
```




Отдельно хотелось бы показать описание того, как использовать downwardAPI в 2 форматах — как переменные окружения и как Volume. downwardAPI — эти тип Volume, который дает монтировать значения метаданных пода внутри контейнера в виде файлов.

```
apiVersion: v1
kind: Pod
metadata:
  name: downward
spec:
  containers:
  - name: main
```

```
    image: busybox
    command: ["sleep", "9999999"]
    resources:
      requests:
        cpu: 15m
        memory: 100Ki
      limits:
        cpu: 100m
        memory: 4Mi
    env:
    - name: POD_NAME
      valueFrom:
        fieldRef:
          fieldPath: metadata.name
    - name: POD_NAMESPACE
      valueFrom:
        fieldRef:
          fieldPath: metadata.namespace
    - name: POD_IP
      valueFrom:
        fieldRef:
          fieldPath: status.podIP
    - name: NODE_NAME
      valueFrom:
        fieldRef:
          fieldPath: spec.nodeName
    - name: SERVICE_ACCOUNT
      valueFrom:
        fieldRef:
          fieldPath: spec.serviceAccountName
    - name: CONTAINER_CPU_REQUEST_MILLICORES
      valueFrom:
        resourceFieldRef:
          resource: requests.cpu
          divisor: 1m
    - name: CONTAINER_MEMORY_LIMIT_KIBIBYTES
      valueFrom:
        resourceFieldRef:
          resource: limits.memory
          divisor: 1Ki
---
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: downward
  labels:
    foo: bar
  annotations:
    key1: value1
    key2: |
      multi
      line
      value
spec:
  containers:
  - name: main
    image: busybox
    command: ["sleep", "9999999"]
    resources:
      requests:
        cpu: 15m
        memory: 100Ki
      limits:
        cpu: 100m
        memory: 4Mi
    volumeMounts:
    - name: downward
      mountPath: /etc/downward
  volumes:
  - name: downward
    downwardAPI:
      items:
      - path: "podName"
        fieldRef:
          fieldPath: metadata.name
      - path: "podNamespace"
        fieldRef:
          fieldPath: metadata.namespace
      - path: "labels"
        fieldRef:
          fieldPath: metadata.labels
      - path: "annotations"
```

```
      fieldRef:
        fieldPath: metadata.annotations
    - path: "containerCpuRequestMilliCores"
      resourceFieldRef:
        containerName: main
        resource: requests.cpu
        divisor: 1m
    - path: "containerMemoryLimitBytes"
      resourceFieldRef:
        containerName: main
        resource: limits.memory
        divisor: 1
```

## Secrets

Секреты почти как configmap также хранят key=value пары значений. Но в отличие от configmap они доставляются исключительно на те ноды, которые запускают поды, использующие эти секреты. Всегда хранятся в памяти и не пишутся на диск. Кроме того, значения в secret хранятся в зашифрованном виде. По умолчанию base64, но есть и другие алгоритмы. Предположим, мы хотим сохранить в секрет пароль от базы данных.

```
kubectl create secret generic my-password
--from-literal=password=mysqlpassword
```

После создания попробуйте прочитать секрет:

```
kubectl get secret my-password
```

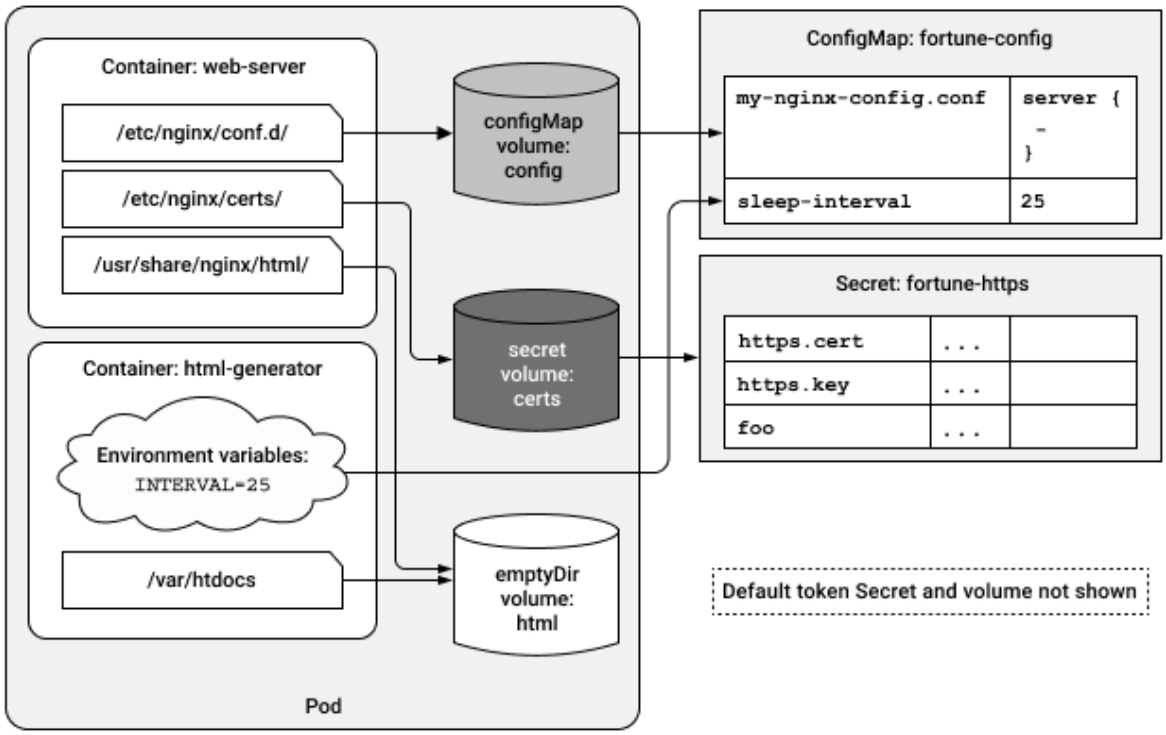
Значение скрыто, но, если добавить -o yaml или json, вы увидите шифрованный секрет. Конечно, base64 не слишком безопасен, есть другие методы, которые куда более применимы. О них можно прочитать в этой статье Encrypting Secret Data at Rest.
Секреты также можно использовать как переменные окружения и файлы.

```
spec:
  containers:
  - image: wordpress:4.7.3-apache
    name: wordpress
    env:
    - name: WORDPRESS_DB_HOST
      value: wordpress-mysql
    - name: WORDPRESS_DB_PASSWORD
      valueFrom:
        secretKeyRef:
          name: my-password
```

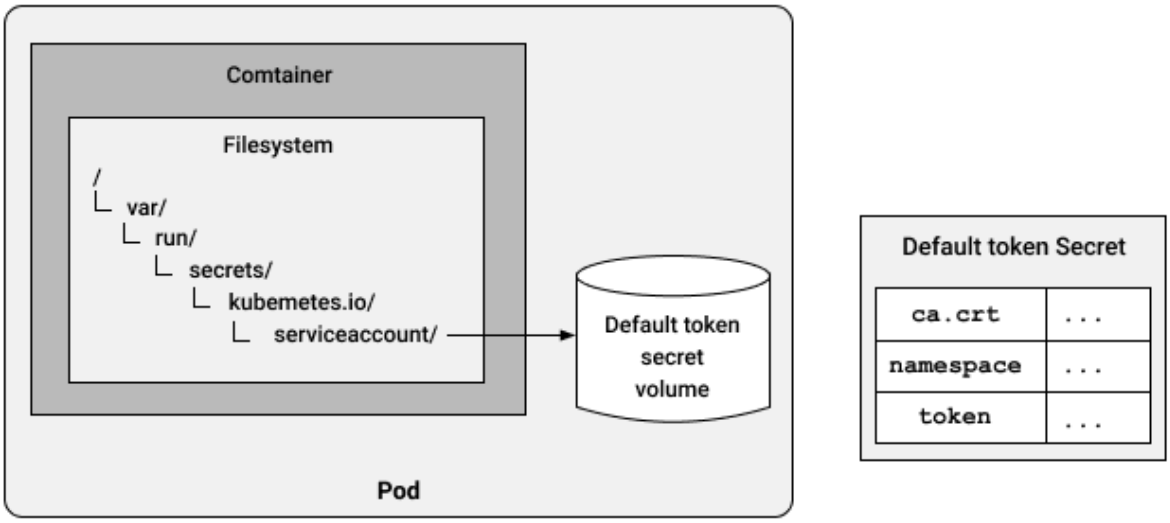
```
          key: password
---
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
  - name: mypod
    image: redis
    volumeMounts:
    - name: foo
      mountPath: "/etc/foo"
      readOnly: true
  volumes:
  - name: foo
    secret:
      secretName: mysecret
```

Если в секрете содержится несколько значений, можно выборочно использовать требуемые:

```
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
  - name: mypod
    image: redis
    volumeMounts:
    - name: foo
      mountPath: "/etc/foo"
      readOnly: true
  volumes:
  - name: foo
    secret:
      secretName: mysecret
      items:
      - key: username
        path: my-group/my-username
```




Кстати, каждый pod по умолчанию получает секрет default-secret, который содержит token для доступа к kubernetes API.




# Полезные ссылки:

- Знакомство с Kubernetes. Secrets
- Знакомство с Kubernetes. ConfigMaps

# Задание:

1. Создайте configmap nginx-conf в namespace default, содержащую конфигурацию для nginx — nginx.conf, в которой указано количество воркеров — 4 штуки.
2. Создайте secret nginx-env в namespace default с переменной AUTH=testpassword.
3. Создайте под nginx в namespace default, который будет монтировать конфигурацию из configmap и добавлять env переменную из secret.
4. Отправьте задание на проверку.