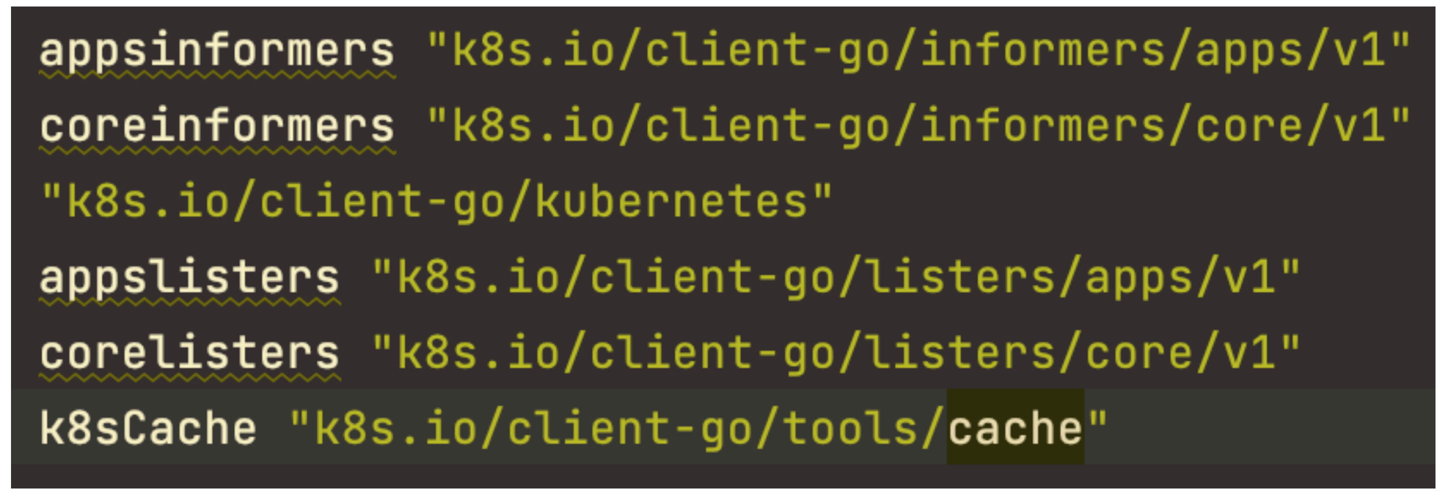
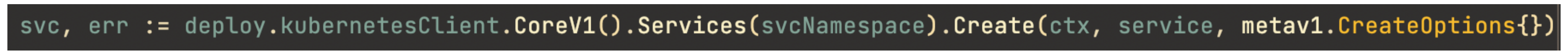
k8s与fission如何结合

使用方法概述

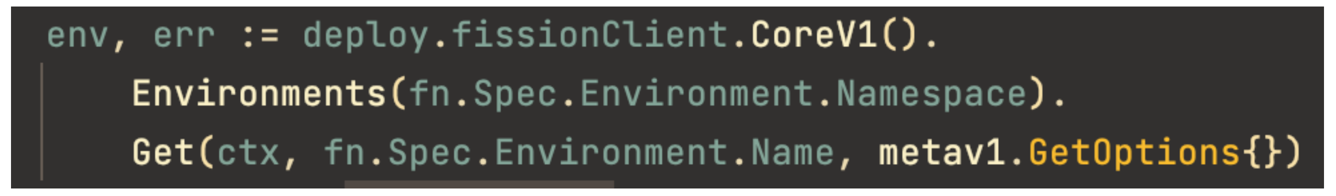
- 调用k8s 提供给go的interface (api) -- **client go** library
    - The client-go library is an official kubernetes client SDK by K8s community, you can use this library to programmatically **manipulate your kubernetes cluster**
    - The kubernetes client tool **kubectl** is also built using client-go
    - kubenets - contains the **clientset** to access **Kubernetes API**
- 使用方法
    - import library
    - 代码，像kubectl部署k8s
        - kubectl create service ...
    - 运行k8s
    - 下载lib，编译以后运行，api生效
- Fission client library
    - fission提供给自身使用的接口
    - fission environment create —name xxx —image



```
appsinformers "k8s.io/client-go/informers/apps/v1"
coreinformers "k8s.io/client-go/informers/core/v1"
"k8s.io/client-go/kubernetes"
appslisters "k8s.io/client-go/listers/apps/v1"
corelisters "k8s.io/client-go/listers/core/v1"
k8sCache "k8s.io/client-go/tools/cache"
```



```
svc, err := deploy.kubernetesClient.CoreV1().Services(svcNamespace).Create(ctx, service, metav1.CreateOptions{})
```

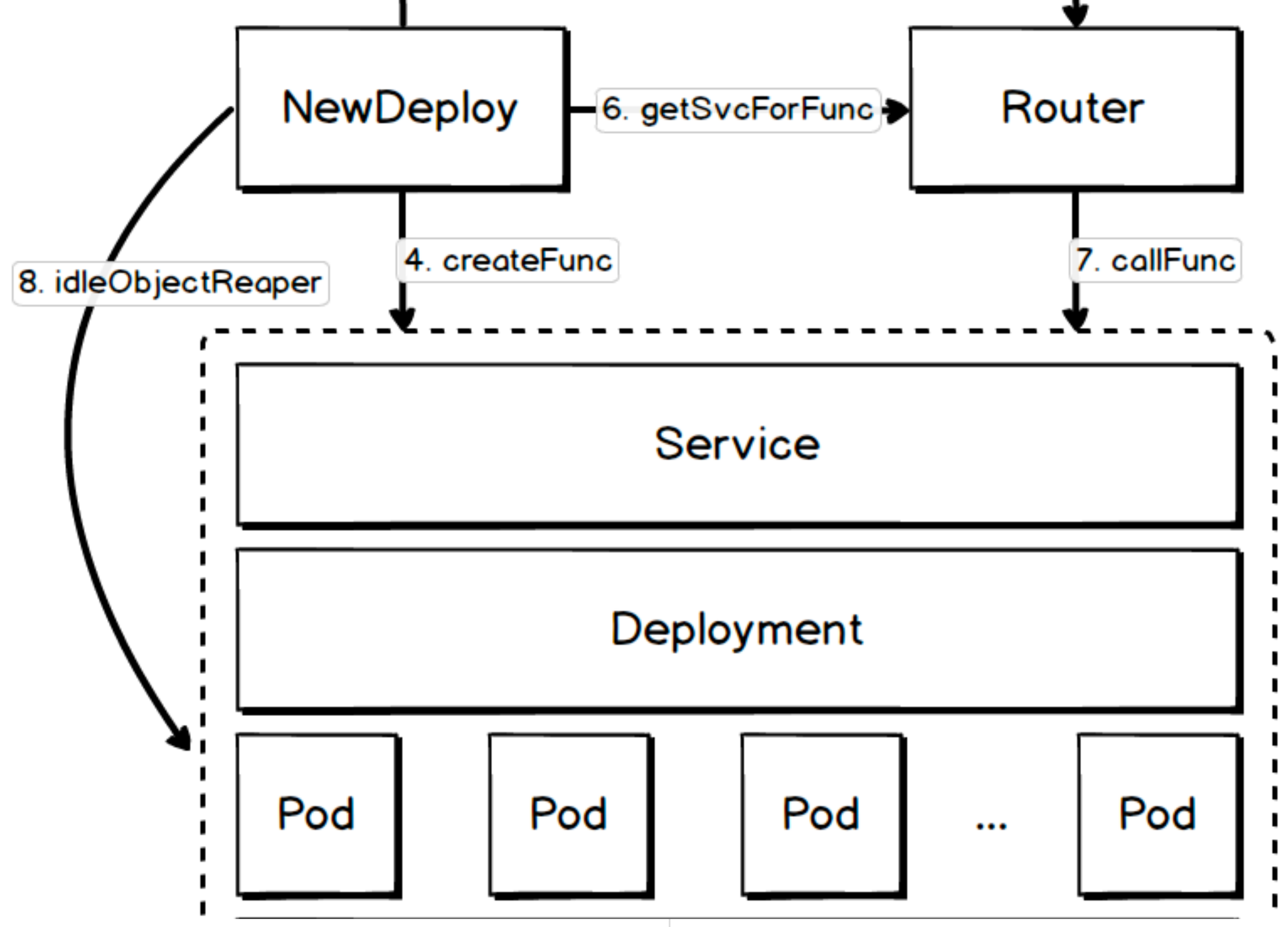


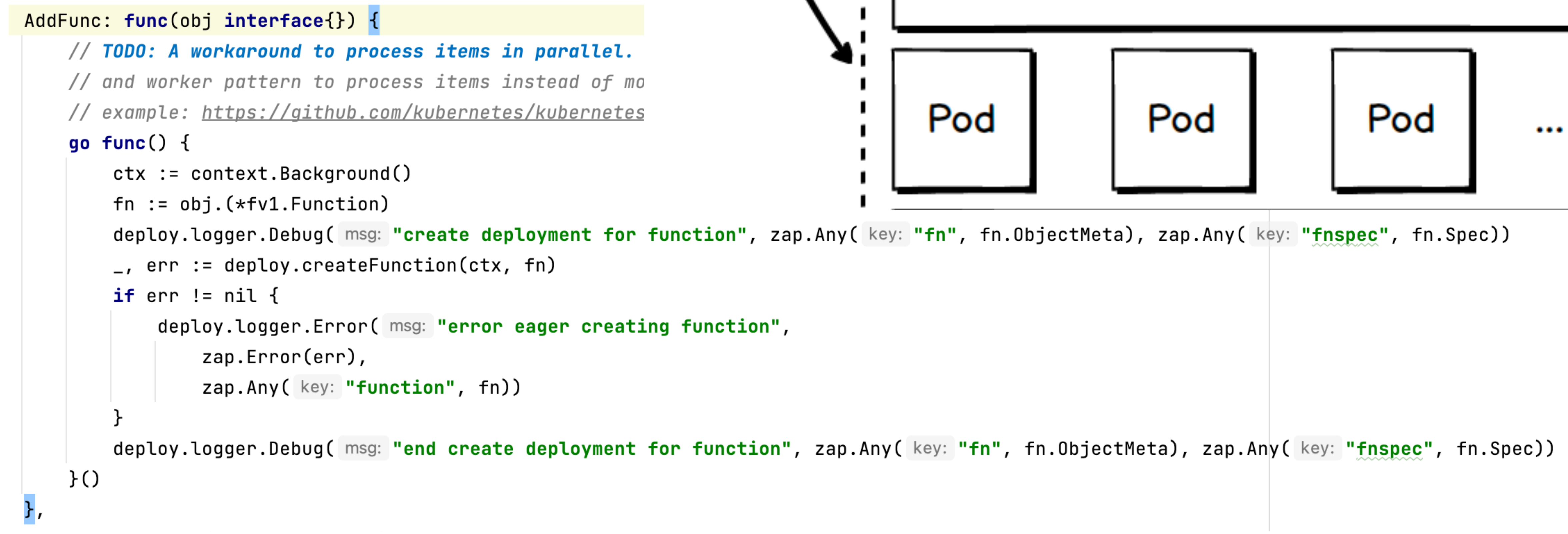
```
env, err := deploy.fissionClient.CoreV1().
    Environments(fn.Spec.Environment.Namespace).
    Get(ctx, fn.Spec.Environment.Name, metav1.GetOptions{})
```

k8s与fission如何结合

CreateFunc代码细节

- AddFunc
  - createFunc
- createFunction
  - Parameter: context, function (struct)
  - Return: **function service**
  - 调用fnCreate





```
AddFunc: func(obj interface{}) {
    // TODO: A workaround to process items in parallel.
    // and worker pattern to process items instead of mo
    // example: https://github.com/kubernetes/kubernetes
    go func() {
        ctx := context.Background()
        fn := obj.(*fv1.Function)
        deploy.logger.Debug( msg: "create deployment for function", zap.Any( key: "fn", fn.ObjectMeta), zap.Any( key: "fnspec", fn.Spec))
        _, err := deploy.createFunction(ctx, fn)
        if err != nil {
            deploy.logger.Error( msg: "error eager creating function",
                zap.Error(err),
                zap.Any( key: "function", fn))
        }
        deploy.logger.Debug( msg: "end create deployment for function", zap.Any( key: "fn", fn.ObjectMeta), zap.Any( key: "fnspec", fn.Spec))
    }()
},
```

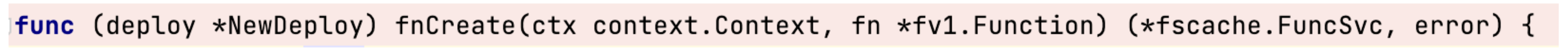
```
func (deploy *NewDeploy) createFunction(ctx context.Context, fn *fv1.Function) (*fscache.FuncSvc, error)
```

```
func (deploy *NewDeploy) fnCreate(ctx context.Context, fn *fv1.Function) (*fscache.FuncSvc, error) {
```

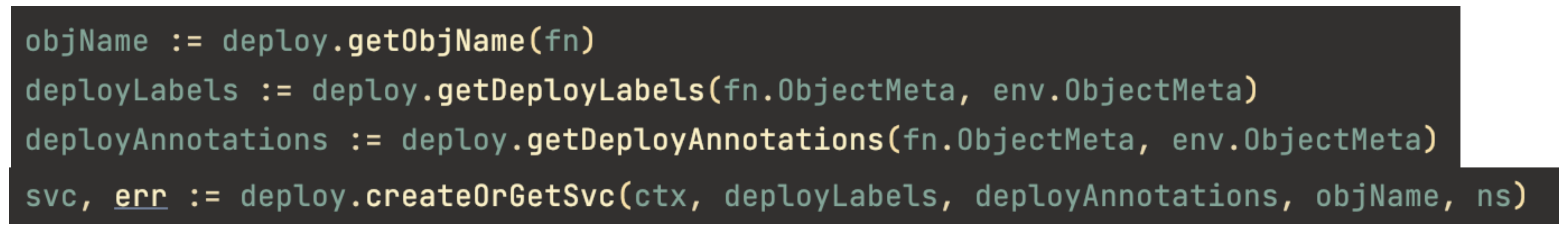
k8s与fission如何结合

CreateFunc代码细节

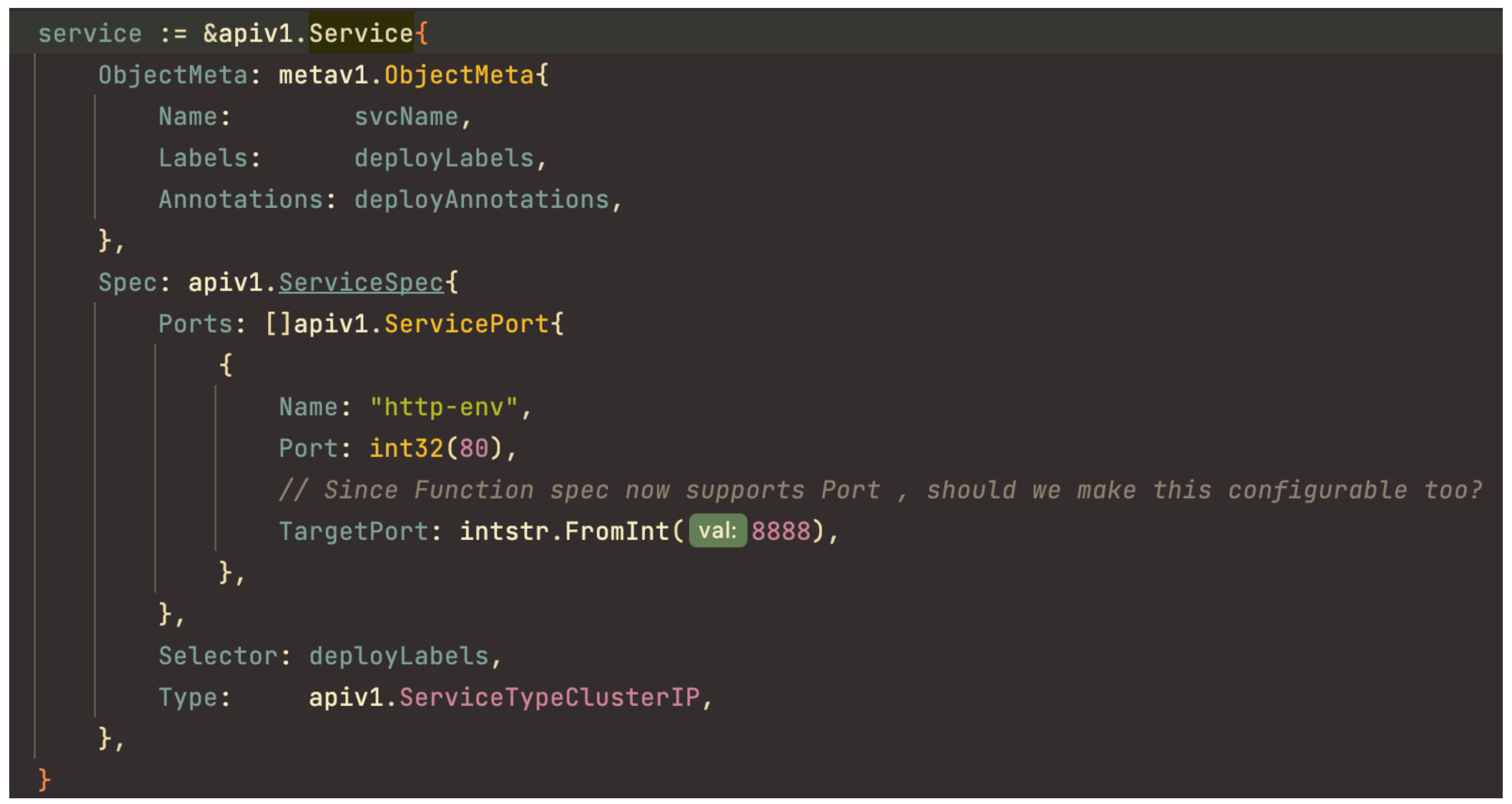
- fnCreate
  - **input:** context, function; **return:** function service
  - createOrGetSvc
  - createOrGetDeployment
  - 封装fsvc
- createOrGetSvc
  - **input**: context, deploylabels, objName, deployAnnotations; **return**: service
  - 提供类似service.yaml配置文件中的信息，用来创建service
  - 利用kubenetsClient api



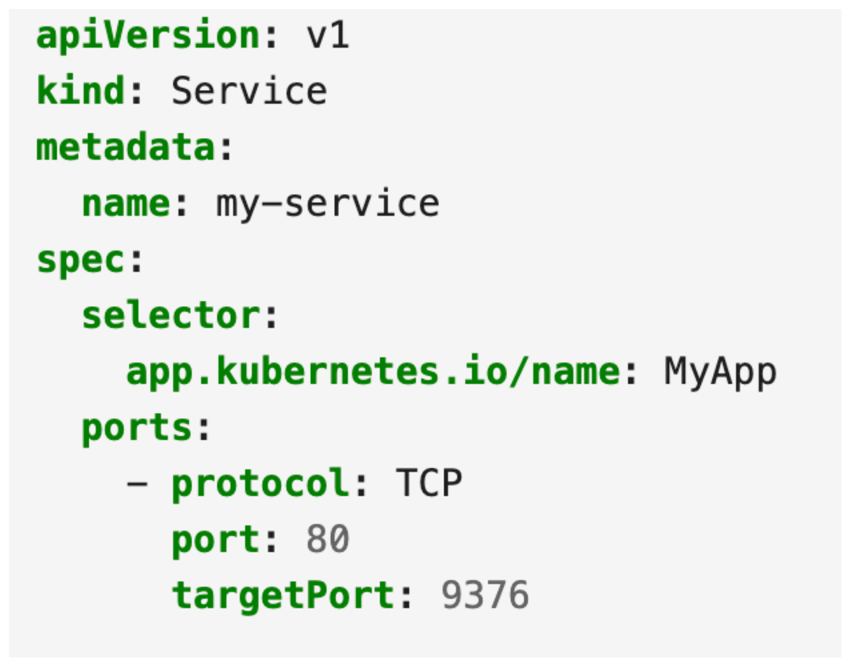
```go
func (deploy *NewDeploy) fnCreate(ctx context.Context, fn *fv1.Function) (*fscache.FuncSvc, error) {
```



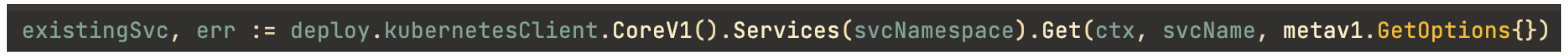
```go
objName := deploy.getObjName(fn)
deployLabels := deploy.getDeployLabels(fn.ObjectMeta, env.ObjectMeta)
deployAnnotations := deploy.getDeployAnnotations(fn.ObjectMeta, env.ObjectMeta)
svc, err := deploy.createOrGetSvc(ctx, deployLabels, deployAnnotations, objName, ns)
```



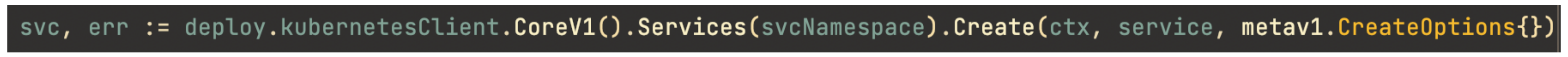
```go
service := &apiv1.Service{
	ObjectMeta: metav1.ObjectMeta{
		Name:        svcName,
		Labels:      deployLabels,
		Annotations: deployAnnotations,
	},
	Spec: apiv1.ServiceSpec{
		Ports: []apiv1.ServicePort{
			{
				Name: "http-env",
				Port: int32(80),
				// Since Function spec now supports Port , should we make this configurable too?
				TargetPort: intstr.FromInt( val: 8888),
			},
		},
		Selector: deployLabels,
		Type:     apiv1.ServiceTypeClusterIP,
	},
}
```



```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app.kubernetes.io/name: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```



```go
existingSvc, err := deploy.kubernetesClient.CoreV1().Services(svcNamespace).Get(ctx, svcName, metav1.GetOptions{})
```

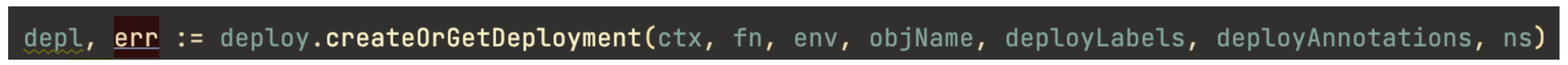


```go
svc, err := deploy.kubernetesClient.CoreV1().Services(svcNamespace).Create(ctx, service, metav1.CreateOptions{})
```

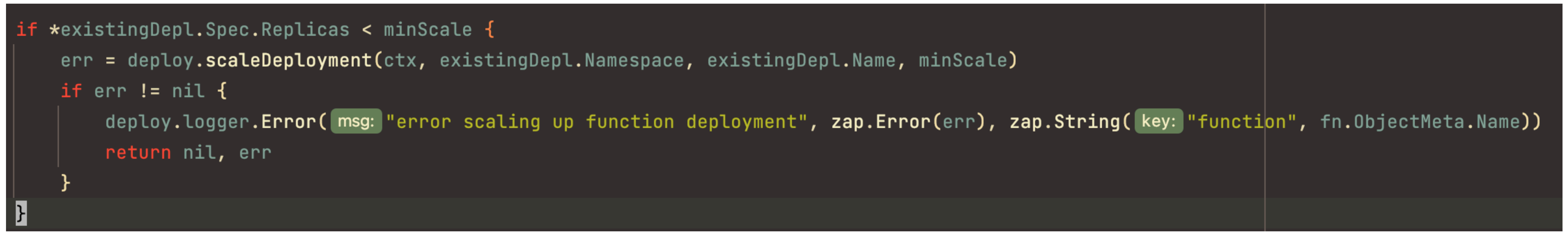
k8s与fission如何结合

CreateFunc代码细节

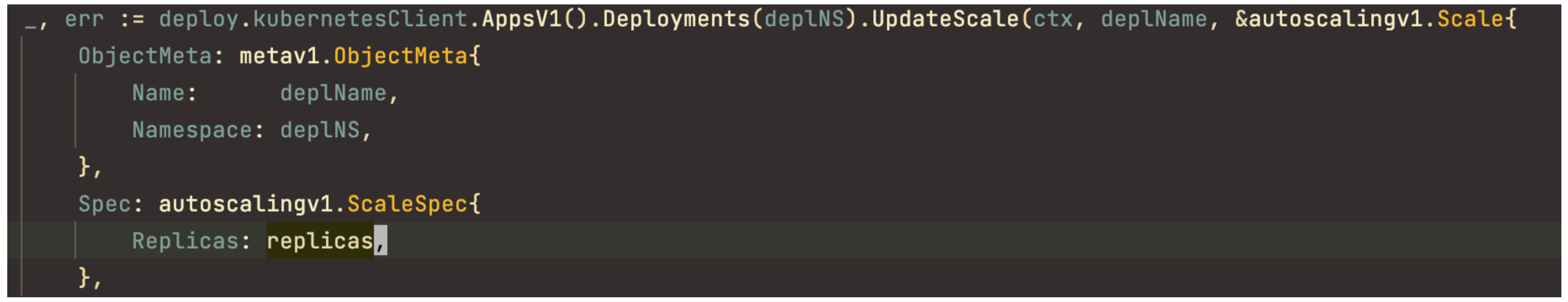
- createOrGetDeployment
  - **input**: fn, env ...; return: deployment
  - 利用k8s api，并实现newdeploy的策略
    - 比如扩展replica数量



```go
depl, err := deploy.createOrGetDeployment(ctx, fn, env, objName, deployLabels, deployAnnotations, ns)
```



```go
if *existingDepl.Spec.Replicas < minScale {
    err = deploy.scaleDeployment(ctx, existingDepl.Namespace, existingDepl.Name, minScale)
    if err != nil {
        deploy.logger.Error(msg: "error scaling up function deployment", zap.Error(err), zap.String(key: "function", fn.ObjectMeta.Name))
        return nil, err
    }
}
```

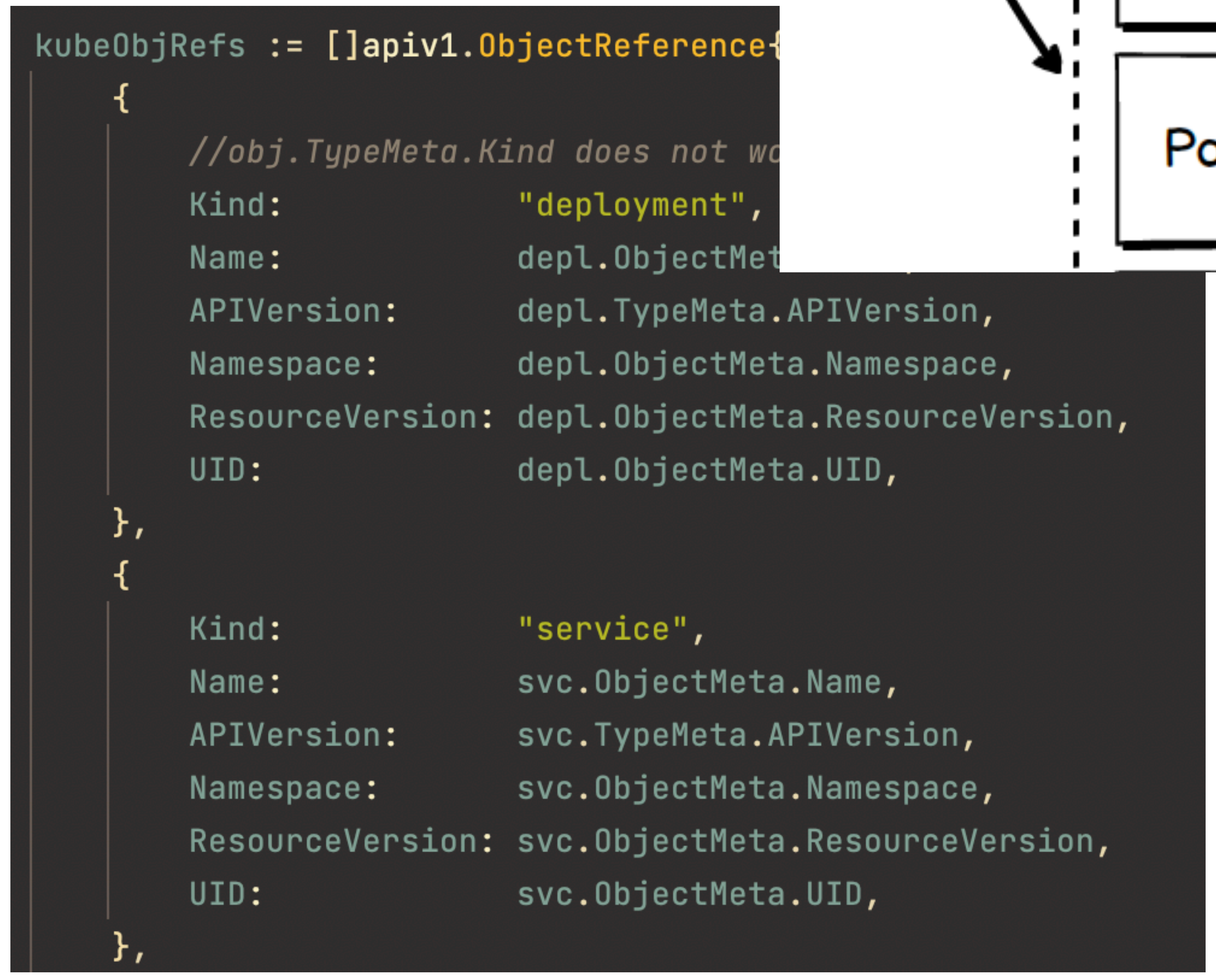


```go
_, err := deploy.kubernetesClient.AppsV1().Deployments(deplNS).UpdateScale(ctx, deplName, &autoscalingv1.Scale{
    ObjectMeta: metav1.ObjectMeta{
        Name:      deplName,
        Namespace: deplNS,
    },
    Spec: autoscalingv1.ScaleSpec{
        Replicas: replicas,
    },
```

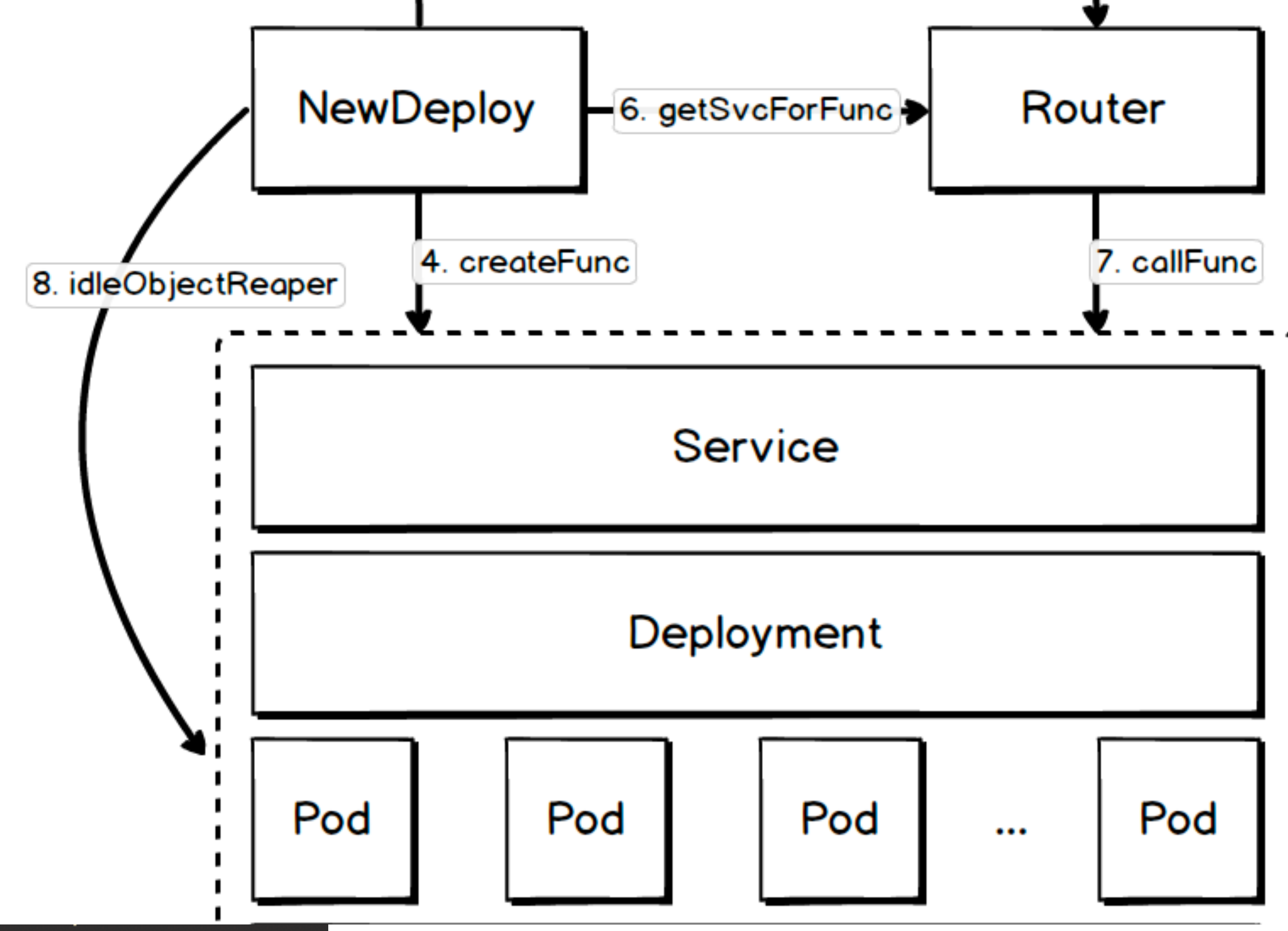
k8s与fission如何结合

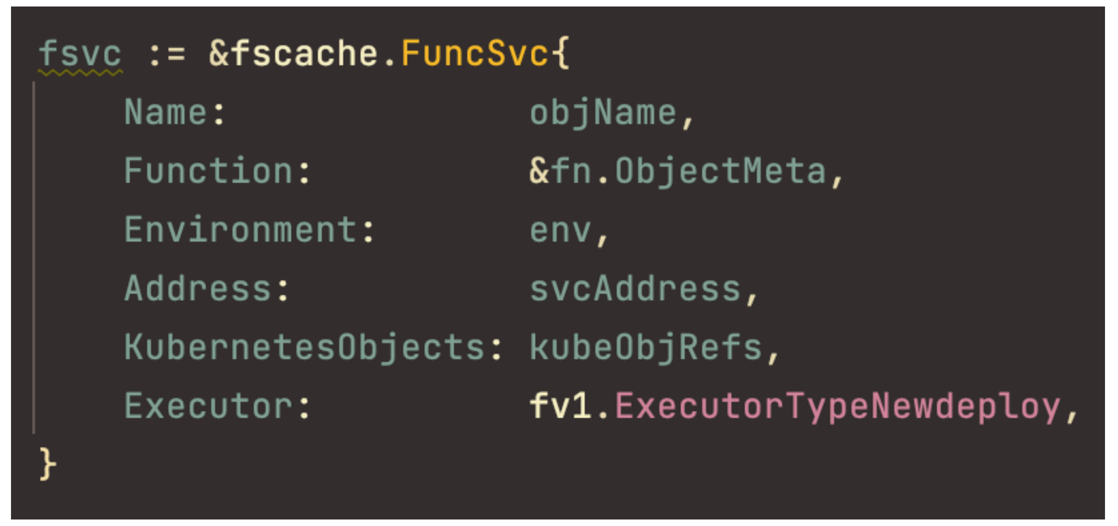
CreateFunc代码细节

- 封装fsvc
    - 封装kube obj
        - depl - deployment
        - service - k8s service
    - 封装function service
        - KubernetesObjects: kube obj
        - address - svc.namespace + svc.name



```
kubeObjRefs := []apiv1.ObjectReference{
    {
        //obj.TypeMeta.Kind does not wo
        Kind:            "deployment",
        Name:            depl.ObjectMet
        APIVersion:      depl.TypeMeta.APIVersion,
        Namespace:       depl.ObjectMeta.Namespace,
        ResourceVersion: depl.ObjectMeta.ResourceVersion,
        UID:             depl.ObjectMeta.UID,
    },
    {
        Kind:            "service",
        Name:            svc.ObjectMeta.Name,
        APIVersion:      svc.TypeMeta.APIVersion,
        Namespace:       svc.ObjectMeta.Namespace,
        ResourceVersion: svc.ObjectMeta.ResourceVersion,
        UID:             svc.ObjectMeta.UID,
    },
```





```
fsvc := &fscache.FuncSvc{
    Name:              objName,
    Function:          &fn.ObjectMeta,
    Environment:       env,
    Address:           svcAddress,
    KubernetesObjects: kubeObjRefs,
    Executor:          fv1.ExecutorTypeNewdeploy,
}
```

wasmedge: run wasm in k8s

- Minikube start —kubernetes-version=v1.23.8
  - kubernet V1.24及以后可能有点问题（kubelet）；
- **仍未复现成功**
  - 出现了很多问题
  - 目前crio配置crun完成，还剩下用crun运行wasm文件、k8s运行wasm文件两个关键步骤
  - crio创建pod出现问题

参考

- Create Kubernetes Jobs in Golang using K8s client-go API
- WasmEdge in Kubernetes