

从Docker到Kubernetes 第14周

DATAGURU专业数据分析社区

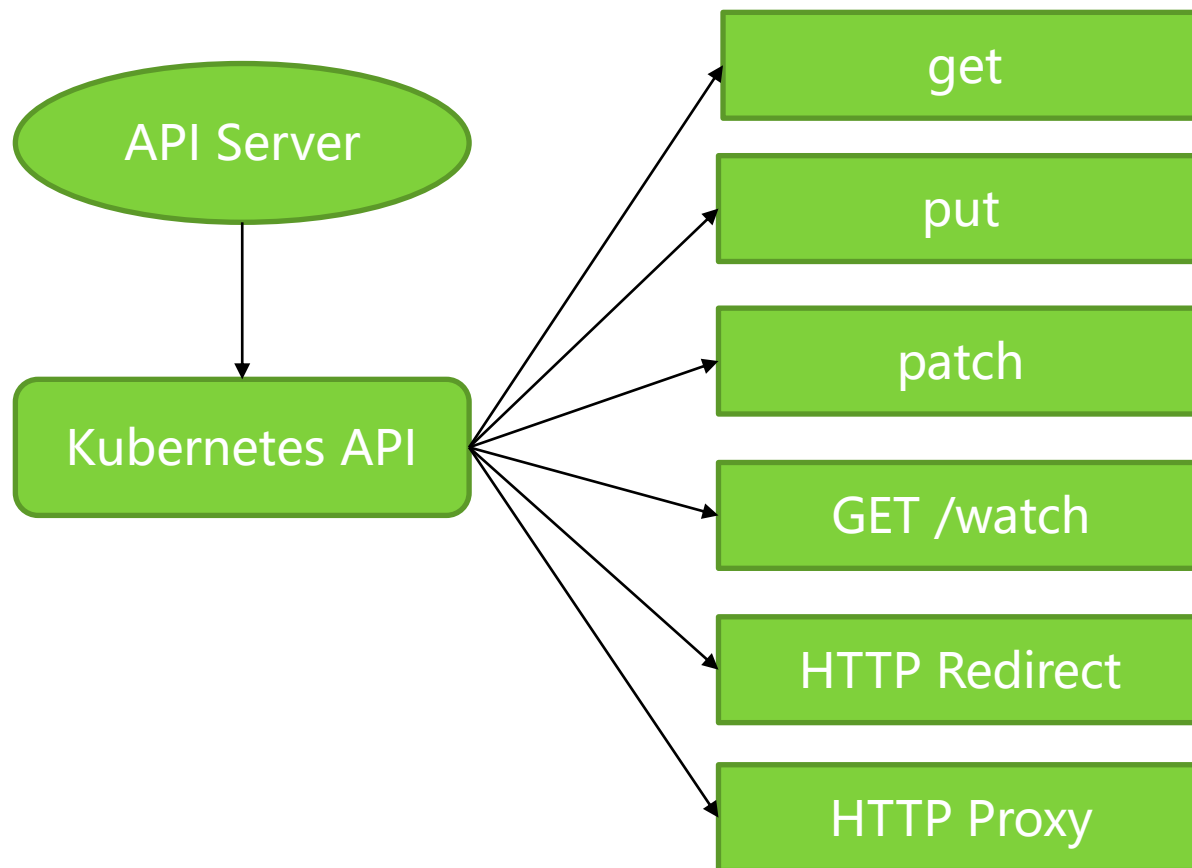
【声明】 本视频和幻灯片为炼数成金网络课程的教学资料，所有资料只能在课程内使用，不得在课程以外范围散播，违者将可能被追究法律和经济责任。

课程详情访问炼数成金培训网站

<http://edu.dataguru.cn>

- Kubernetes API入门
- Kubernetes 源码入门
- Ku8 eye开源项目

Kubernetes Rest API的接口形式



Kubernetes API入门



Kubernetes Rest API汇总 (一)

资源类型	方法	URL Path	说明	备注
NODES	GET	/api/v1/nodes	获取Node列表	
	POST	/api/v1/nodes	创建一个Node对象	
	DELETE	/api/v1/nodes/{name}	删除一个Node对象	
	GET	/api/v1/nodes/{name}	获取一个Node对象	
	PATCH	/api/v1/nodes/{name}	部分更新一个Node对象	
	PUT	/api/v1/nodes/{name}	替换一个Node对象	
NAMESPACES	GET	/api/v1/namespaces	获得Namespace列表	
	POST	/api/v1/namespaces	创建一个Namespace对象	
	DELETE	/api/v1/namespaces/{name}	删除一个Namespace对象	
	GET	/api/v1/namespaces/{name}	获取一个Namespace对象	
	PATCH	/api/v1/namespaces/{name}	部分更新一个Namespace对象	
	PUT	/api/v1/namespaces/{name}	替换一个Namespace对象	
	PUT	/api/v1/namespaces/{name}/finalize	替换一个Namespace对象的最终方案对象	在 Fabric8 中没有实现
	PUT	/api/v1/namespaces/{name}/status	替换一个Namespace对象的状态	在 Fabric8 中没有实现

Kubernetes API入门

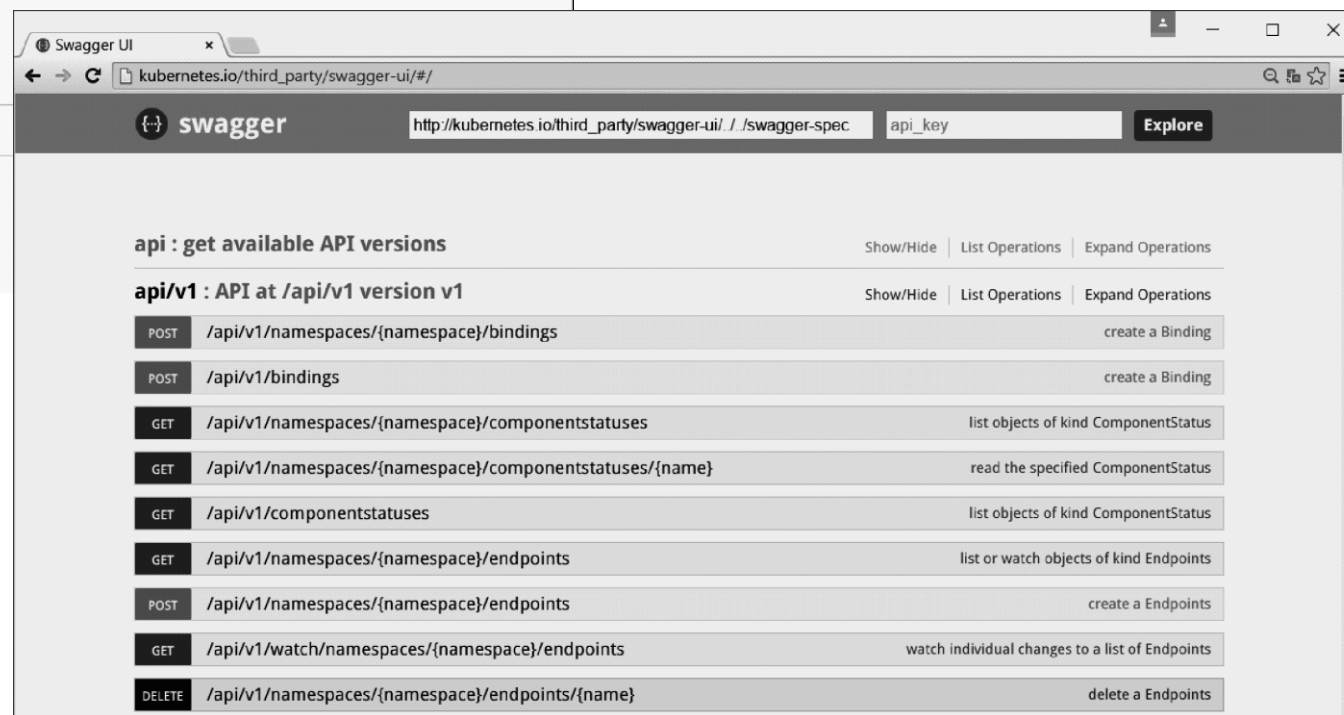
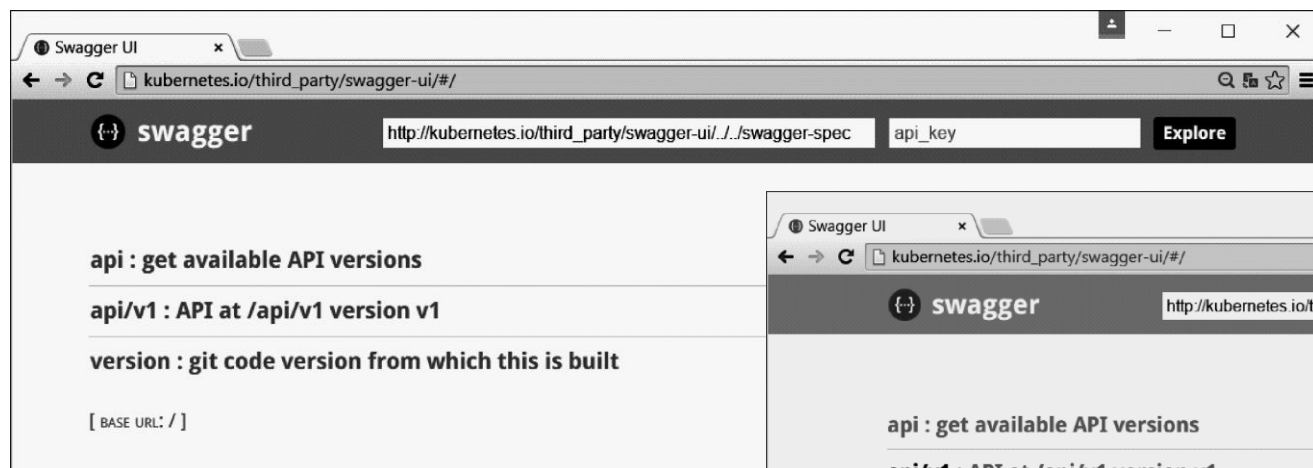


Kubernetes Rest API汇总 (二)

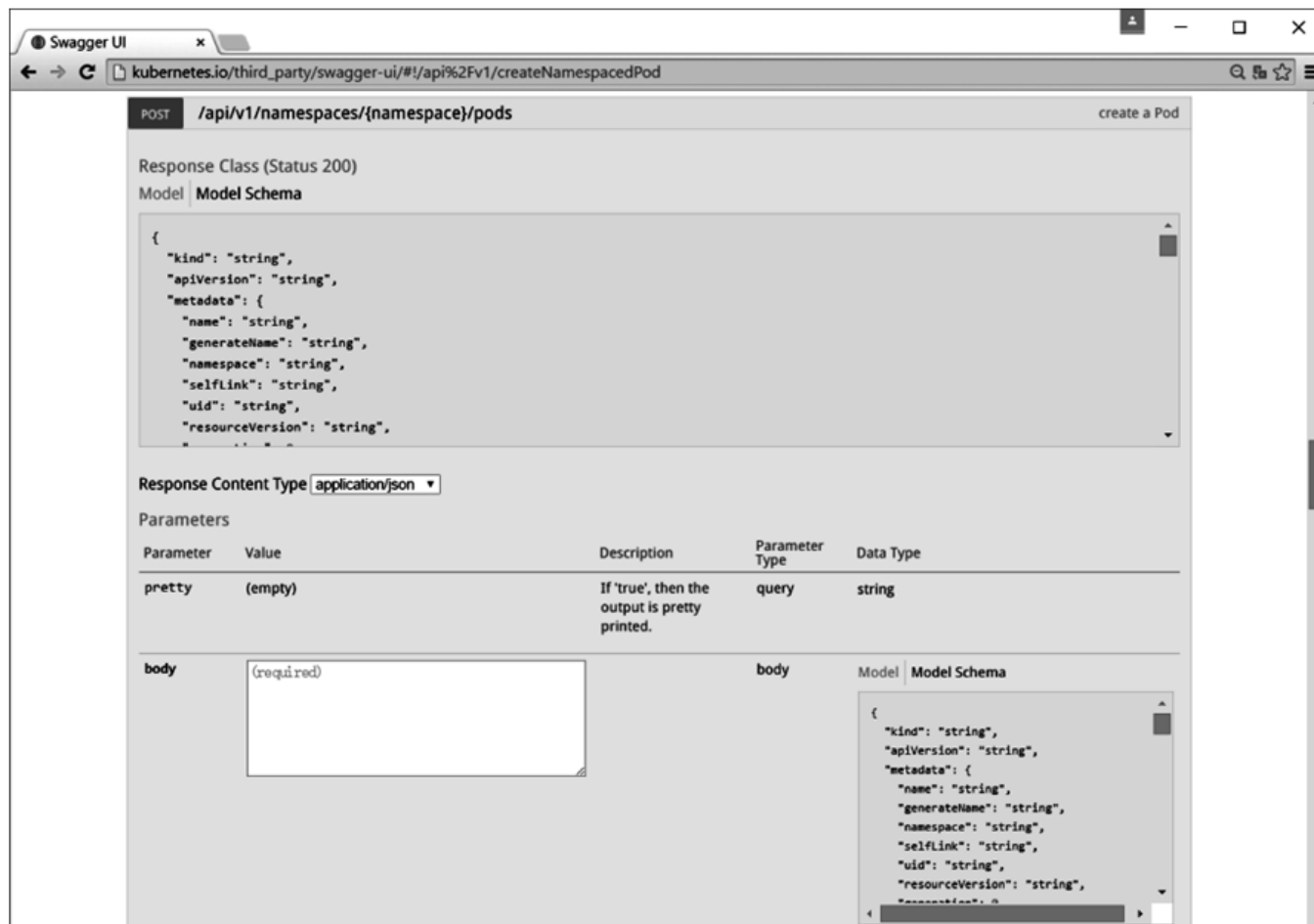
资源类型	方法	URL Path	说明	备注
SERVICES	GET	/api/v1/services	获取Service列表	
	POST	/api/v1/services	创建一个Service对象	
	GET	/api/v1/namespaces/{namespace}/services	获取某个Namespace下的Service列表	
	POST	/api/v1/namespaces/{namespace}/services	在某个Namespace下创建列表	
	DELETE	/api/v1/namespaces/{namespace}/services/{name}	删除某个Namespace的一个Service对象	
	GET	/api/v1/namespaces/{namespace}/services/{name}	获取某个Namespace下的一个Service对象	
	PATCH	/api/v1/namespaces/{namespace}/services/{name}	部分更新某个Namespace下的一个Service对象	
	PUT	/api/v1/namespaces/{namespace}/services/{name}	替换某个Namespace下的一个Service对象	
REPLICATIONCONTROLLERS	GET	/api/v1/replicationcontrollers	获取RC列表	
	POST	/api/v1/replicationcontrollers	创建一个RC对象	
	GET	/api/v1/namespaces/{namespace}/replicationcontrollers	获取某个Namespace下的RC列表	
	POST	/api/v1/namespaces/{namespace}/replicationcontrollers	在某个Namespace下创建一个RC对象	
	DELETE	/api/v1/namespaces/{namespace}/replicationcontrollers/{name}	删除某个Namespace下的RC对象	
	GET	/api/v1/namespaces/{namespace}/replicationcontrollers/{name}	获取某个Namespace下的RC对象	
	PATCH	/api/v1/namespaces/{namespace}/replicationcontrollers/{name}	部分更新某个Namespace下的RC对象	
	PUT	/api/v1/namespaces/{namespace}/replicationcontrollers/{name}	替换某个Namespace下的RC对象	

Kubernetes API入门

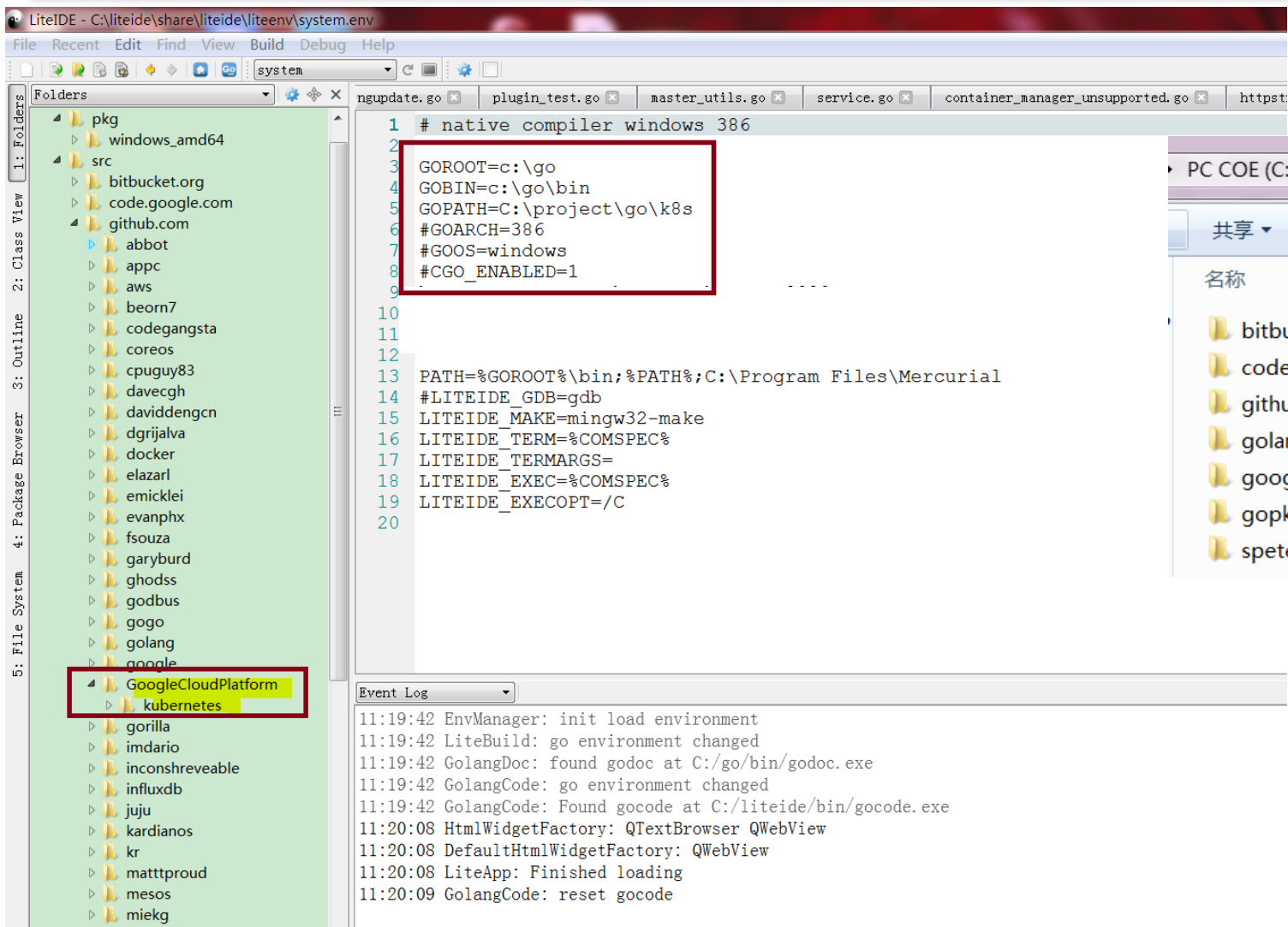
Swagger-UI查看API说明



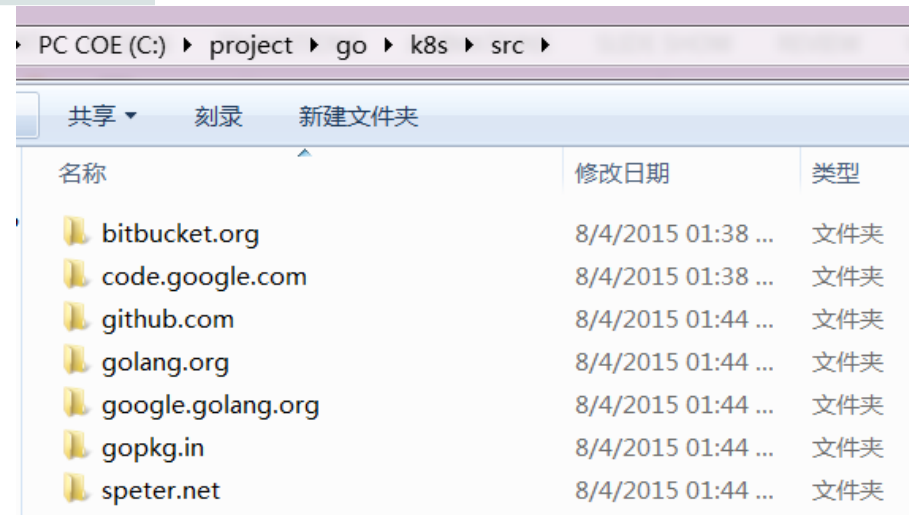
Swagger-UI查看API说明



Kubernetes 源码入门



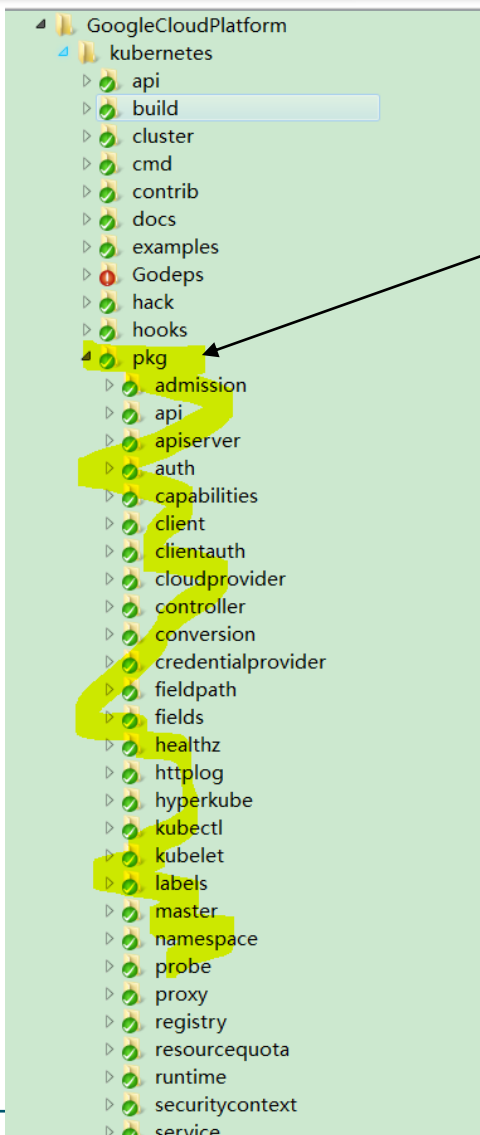
```
1 # native compiler windows 386
2
3 GOROOT=c:\go
4 GOBIN=c:\go\bin
5 GOPATH=C:\project\go\k8s
6 #GOARCH=386
7 #GOOS=windows
8 #CGO_ENABLED=1
9
10
11
12
13 PATH=%GOROOT%\bin;%PATH%;C:\Program Files\Mercurial
14 #LITEIDE_GDB=gdb
15 LITEIDE_MAKE=mingw32-make
16 LITEIDE_TERM=%COMSPEC%
17 LITEIDE_TERMARGS=
18 LITEIDE_EXEC=%COMSPEC%
19 LITEIDE_EXECHOPT=/C
20
```



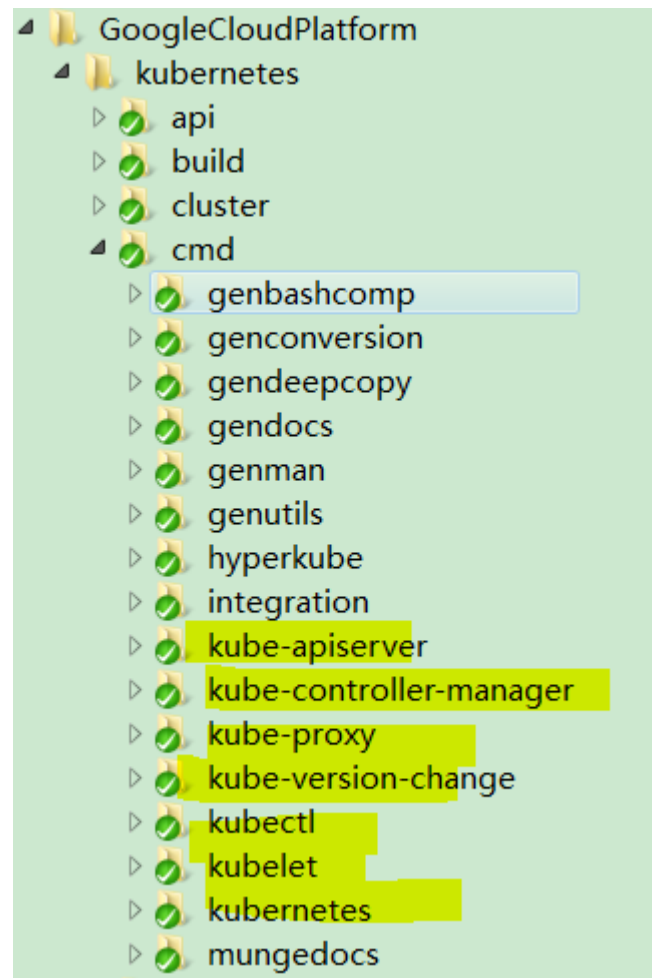
名称	修改日期	类型
bitbucket.org	8/4/2015 01:38 ...	文件夹
code.google.com	8/4/2015 01:38 ...	文件夹
github.com	8/4/2015 01:44 ...	文件夹
golang.org	8/4/2015 01:44 ...	文件夹
google.golang.org	8/4/2015 01:44 ...	文件夹
gopkg.in	8/4/2015 01:44 ...	文件夹
speter.net	8/4/2015 01:44 ...	文件夹

Kubernetes 源码入门

cmd目录下是k8s各个进程的启动进程源码



源码主要在pkg目录下



kube-apiserver进程的入口类名源码位置如下：

github.com/GoogleCloudPlatform/kubernetes/cmd/kube-apiserver/apiserver.go

入口main()函数的逻辑如下：

```
func main() {
    runtime.GOMAXPROCS(runtime.NumCPU())
    rand.Seed(time.Now().UTC().UnixNano())
    s := app.NewAPIServer()
    s.AddFlags(pflag.CommandLine)
    util.InitFlags()
    util.InitLogs()
    defer util.FlushLogs()
    verflag.PrintAndExitIfRequested()

    if err := s.Run(pflag.CommandLine.Args()); err != nil {
        fmt.Fprintf(os.Stderr, "%v\n", err)
        os.Exit(1)
    }
}
```

上述代码核心为下面三行，创建一个APIServer结构体并将命令行启动参数传入，最后启动监听：

```
s := app.NewAPIServer()
s.AddFlags(pflag.CommandLine)
s.Run(pflag.CommandLine.Args())
```



API Server

kube-apiserver进程的启动过程进行了详细分析，我们发现Kubernetes API Service的关键代码就隐藏在pkg\master\master.go里，APIServer这个结构体只不过是一个参数传递通道而已，它的数据最终传给了pkg/master/master.go里的Master结构体

```
// Master contains state for a Kubernetes cluster master/api server.
```

```
type Master struct {
```

```
    // "Inputs", Copied from Config
```

```
    serviceClusterIPRange *net.IPNet
```

```
    serviceNodePortRange util.PortRange
```

```
    cacheTimeout          time.Duration
```

```
    minRequestTimeout     time.Duration
```

```
    mux                    apiserver.Mux
```

```
    muxHelper              *apiserver.MuxHelper
```

```
    handlerContainer       *restful.Container
```

```
    rootWebService         *restful.WebService
```

```
    enableCoreControllers bool
```

```
    enableLogsSupport     bool
```

```
    enableUISupport       bool
```

```
    enableSwaggerSupport  bool
```

```
    enableProfiling       bool
```

```
    apiPrefix             string
```

```
    corsAllowedOriginList util.StringList
```

```
    authenticator          authenticator.Request
```

```
    authorizer             authorizer.Authorizer
```

```
    admissionControl       admission.Interface
```

```
    masterCount            int
```

```
    v1beta3                bool
```

```
    v1                     bool
```

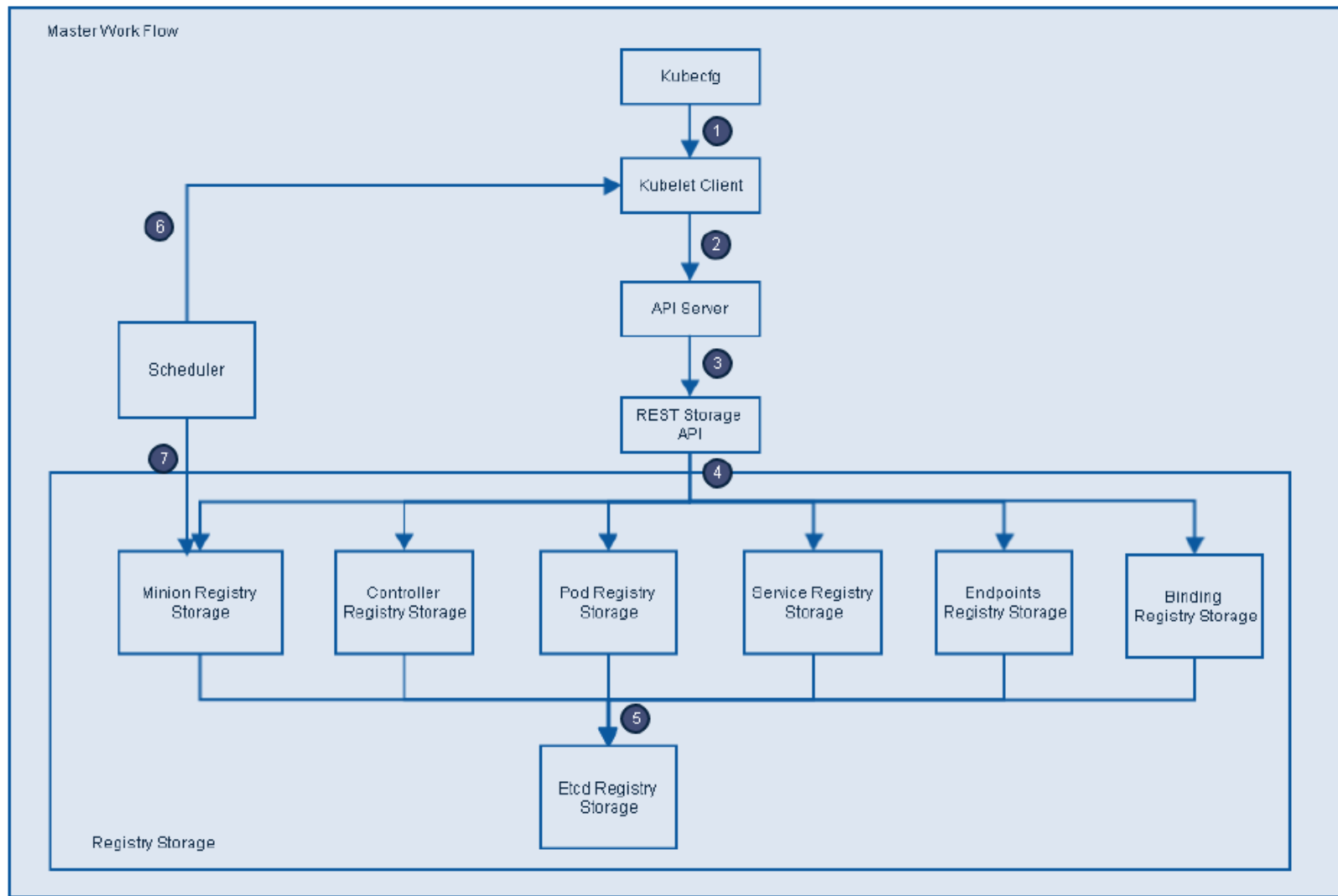
```
    requestContextMapper  api.RequestContextMapper
```

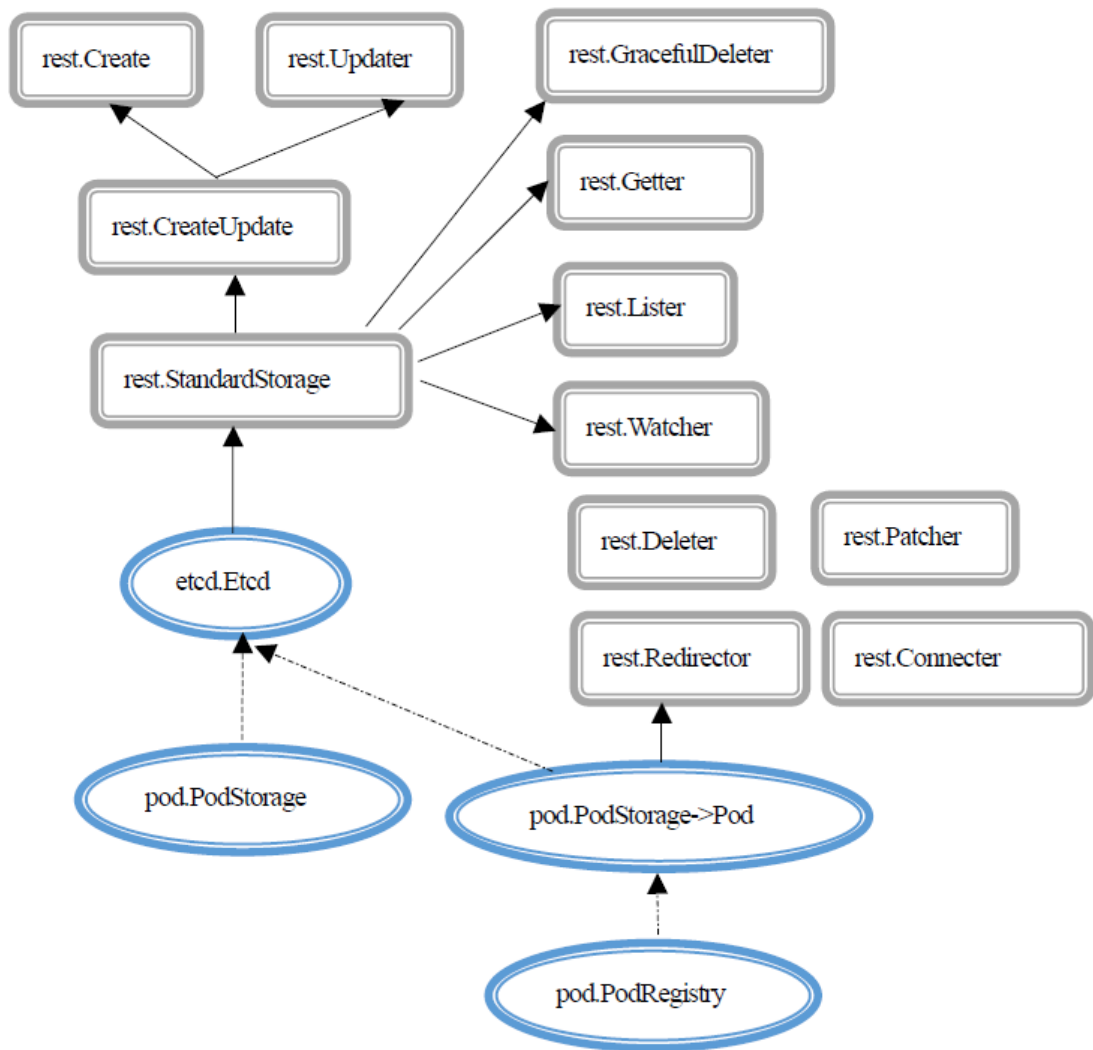
Rest server相关类，采用了go-restful框架设计

权限以及准入控制

go-restful 框架中的核心对象如下。

- ⊙ `restful.Container`: 代表一个 HTTP Rest 服务器，包括一组 `restful.WebService` 对象和一个 `http.ServeMux` 对象，使用 `RouteSelector` 进行请求派发；
- ⊙ `restful.WebService`: 表示一个 Rest 服务，由多个 Rest 路由（`restful.Route`）组成，这一组 Rest 路由共享同一个 Root Path；
- ⊙ `restful.Route`: 表示一个 Rest 路由，Rest 路由主要由 Rest Path、HTTP Method、输入输出类型（HTML/JSON）及对应的回调函数 `restful.RouteFunction` 组成；
- ⊙ `restful.RouteFunction`: 一个用于处理具体的 REST 调用的函数接口定义，具体定义为 `type RouteFunction func(*Request, *Response)`。

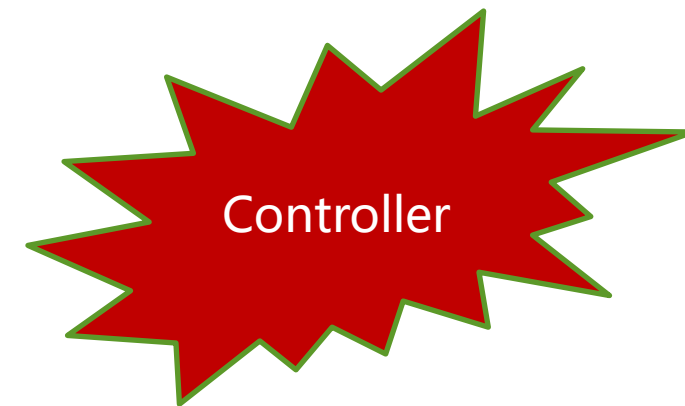
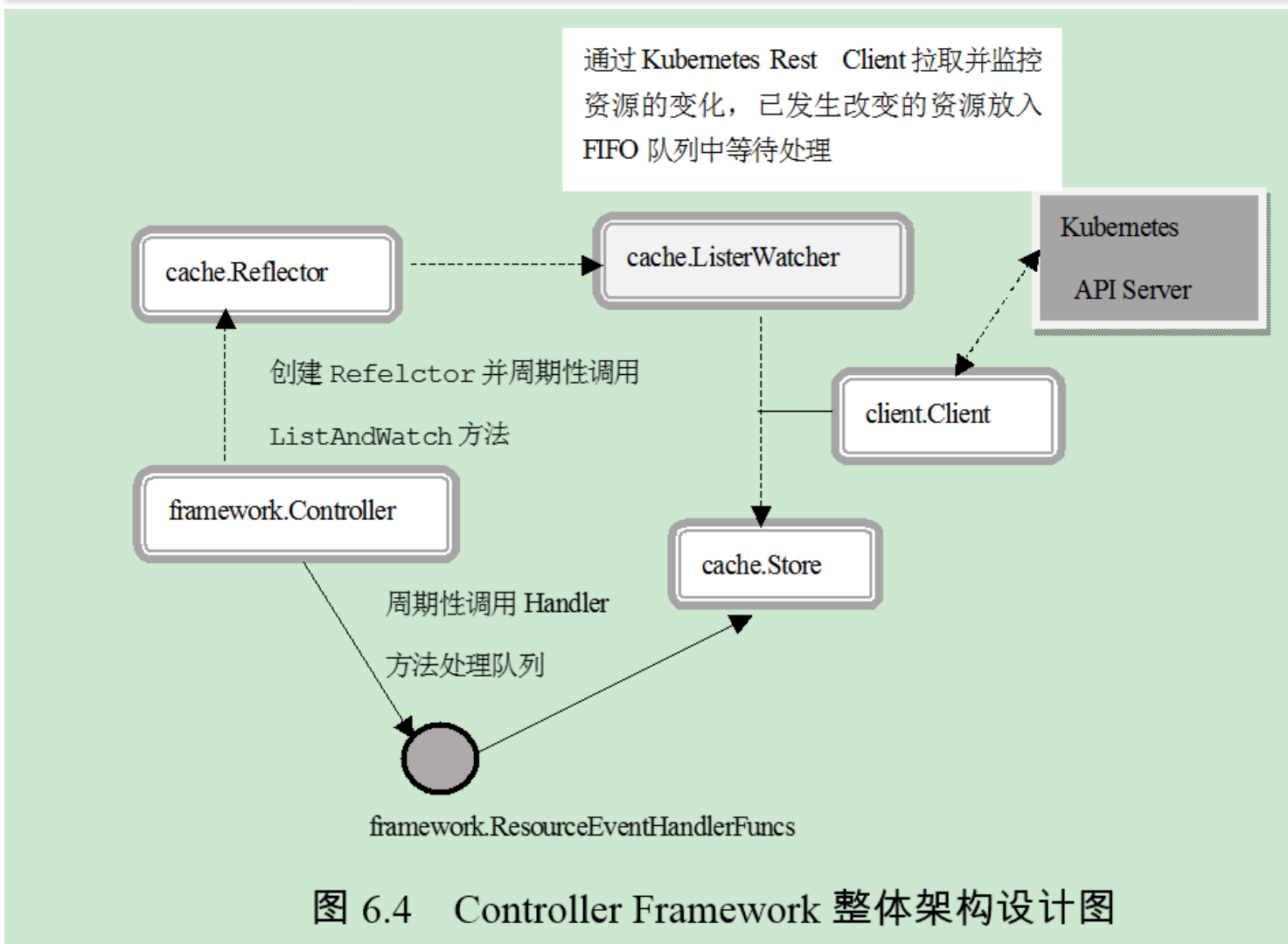




最后，我们来说说 PodRegistry 存在的目的。从之前的代码分析来看，一个来自外部的针对某个资源的 Rest API 发起的请求最后落到对应资源的 rest.Storage 对象上，由 restful.RouteFunction 调用此对象的相关方法完成资源的操作并生成应答返回给客户端，这个过程并没有涉及对应资源的 Registry 服务。那么问题来了，资源的 Registry 接口存在的理由是什么呢？答案很简单，对比 Storage 接口与 Registry 中的资源创建方法的签名，下面是二者的源码对比，后者更符合“手工调用”：

```
Storage 中创建通用的资源对象的接口
Create(ctx api.Context, obj runtime.Object) (runtime.Object, error)
PodRegistry 中创建 Pod 资源的接口
CreatePod(ctx api.Context, pod *api.Pod) error
```

Kubemete API Server 中为每类资源都创建并提供了一个 Registry 接口服务的目的是供内部模块的编程使用，而非对外提供服务，很多文档都错误理解了这个问题。



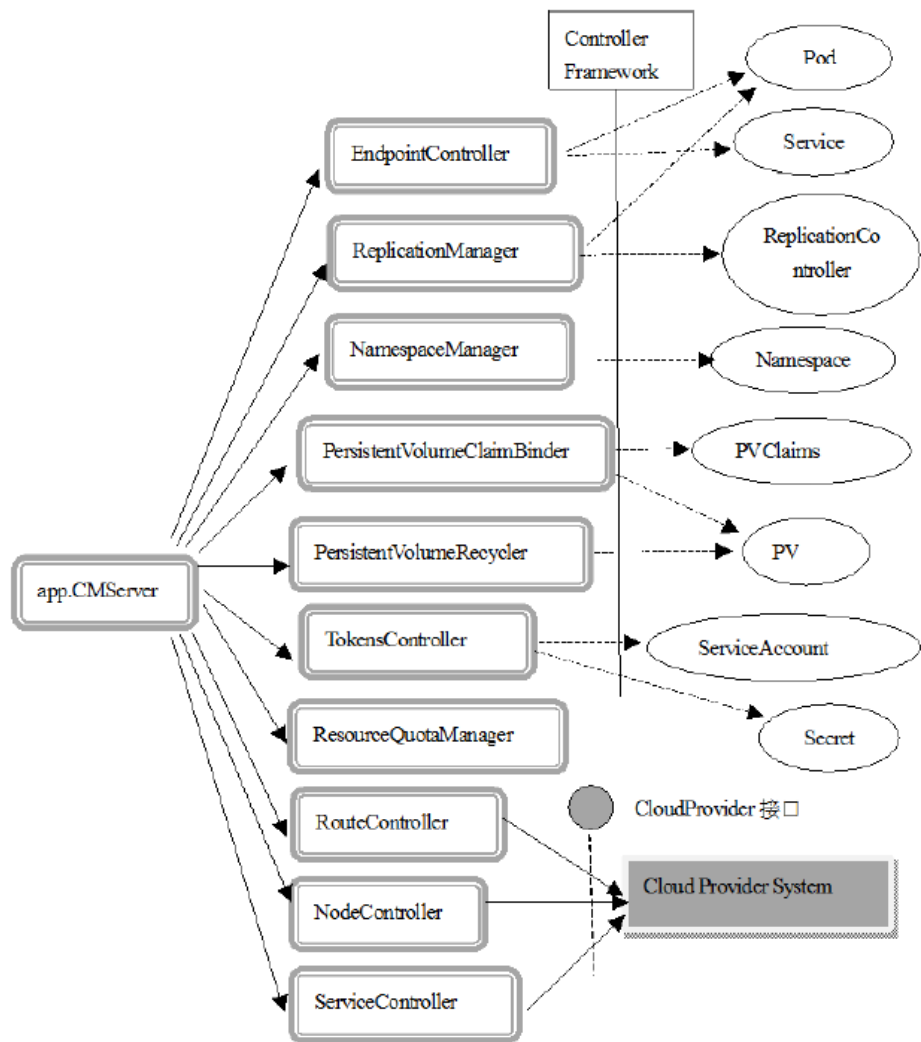


图 6.5 Kubernetes Controller Server 整体设计示意图

为什么会有Ku8 eye开源项目

Kubernetes安装配置
不简单







作为一个分布式平台，
目前没有标准的
“Application”概念
不容易推广使用

Kubernetes还只有容
易出错的命令行操作

K8s eye是一个谷歌Kubernetes的Web一站式管理系统，它具有如下的目标：

- 1.图形化一键安装部署多节点的Kubernetes集群。是安装部署谷歌Kubernetes集群的最快以及最佳方式，安装流程会参考当前系统环境，提供默认优化的集群安装参数，实现最佳部署。
- 2.支持多角色多租户的Portal管理界面。通过一个集中化的Portal界面，运营团队可以很方便的调整集群配置以及管理集群资源，实现跨部门的角色及用户管理、多租户管理，通过自助服务可以很容易完成Kubernetes集群的运维管理工作。
- 3.制定一个Kubernetes应用的程序发布包标准(ku8package)并提供一个向导工具，使得专门为Kubernetes设计的应用能够很容易从本地环境中发布到公有云和其他环境中，更进一步的，我们还提供了Kubernetes应用可视化的构建工具，实现Kubernetes Service、RC、Pod以及其他资源的可视化构建和管理功能
- 4.可定制化的监控和告警系统。内建很多系统健康检查工具用来检测和发现异常并触发告警事件，不仅可以监控集群中的所有节点和组件（包括Docker与Kubernetes），还能够很容易的监控业务应用的性能，我们提供了一个强大的Dashboard，可以用来生成各种复杂的监控图表以展示历史信息，并且可以用来自定义相关监控指标的告警阈值。
- 5.具备的全面的、故障排查能力。平台提供唯一的、集中化的日志管理工具，日志系统从集群中各个节点拉取日志并做聚合分析，拉取的日志包括系统日志和用户程序日志，并且提供全文检索能力以方便故障分析和问题排查，检索的信息包括相关告警信息，而历史视图和相关的度量数据则告诉你，什么时候发生了什么事情，有助于快速了解相关时间内系统的行为特征。
- 6.实现Docker与kubernetes项目的持续集成功能。提供一个可视化工具驱动持续集成的整个流程，包括创建新的Docker镜像、Push镜像到私有仓库中、创建一个Kubernetes测试环境进行测试以及最终滚动升级到生产环境中各个主要环节。

<https://github.com/bestcloud/ku8eye>

 doc	设计文档、用户手册、安装文档等	Merge pull request #76 from gongzh/master
 res	md文档中所引用的图片资源	更新资源管理界面的树
 src	源码工程	Merge pull request #74 from gongzh/master
 Member Profile		Create Member Profile
 README.md		update doc for ku8eye-web:0.3
 github-develop-guide.md		Update github-develop-guide.md

←
Github使用说明，史上最完整的操作文档，建议参与开源的同学都能熟练流程

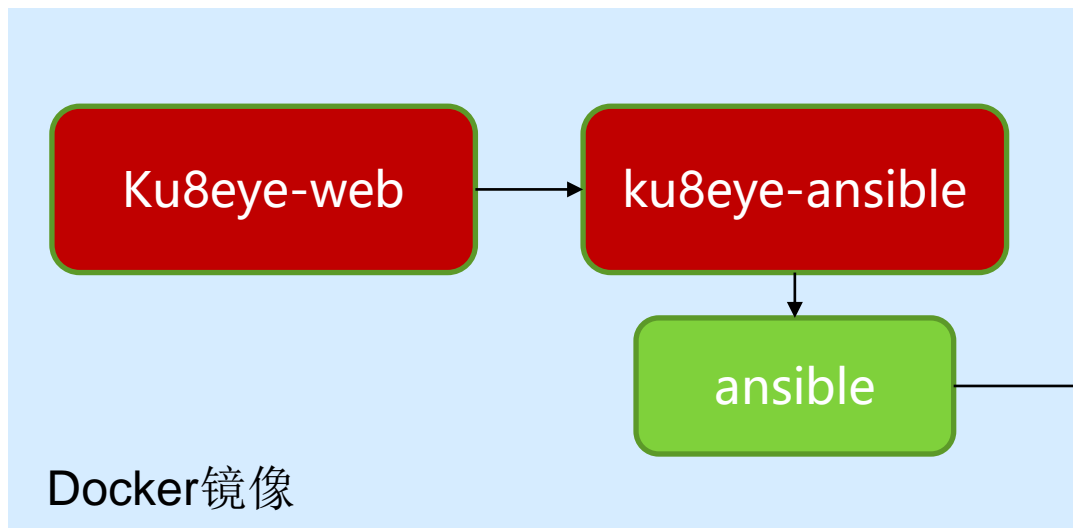
Ku8eye-web采用spring-boot +bootstrap admin模板

src\main目录为主要源码 src\test目录为测试源码，主要为单元测试 src\main目录下如下包结构：

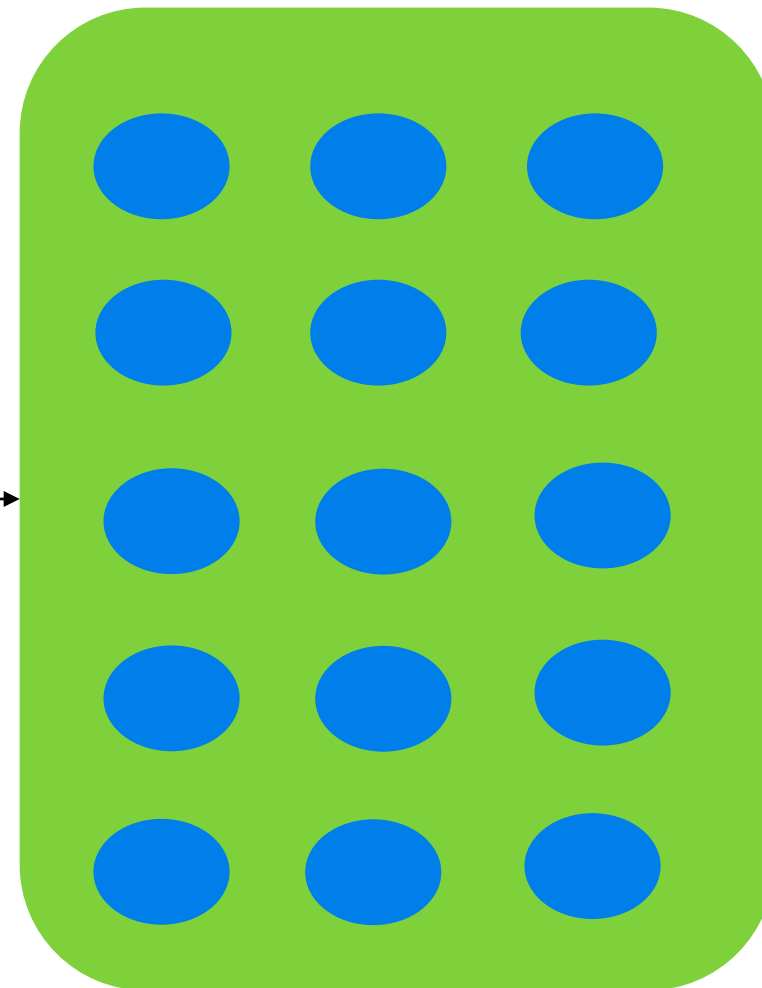
- org.ku8eye.domain目录为存放域对象，这些域对象对应mysql数据库里的一个表
- org.ku8eye.bean，为常规Java Bean对象，用于数据传递或转换等目的
- org.ku8eye.rest，为提供Rest服务的Java服务对象，用于内部或外部系统访问
- org.ku8eye.ctrl，为Spring MVC的Controller对象所在地方
- org.ku8eye.service，为Spring的Service Bean对象所在地方 建议ctrl包与service包可以按照模块名称分子包，比如org.ku8eye.ctrl.user.xxx

页面文件（静态，JSP、JS、Images等）则在以下目录

- src\main\resources\static 在Web里这是ROOT目录



图形化方式一键自
动安装kubernetes
集群



目标物理机

Ku8 eye开源项目

Ku8 eye



guest

● 在线

Search...



主菜单

My Applications

List

Report

K8s Cluster

新增Application

Show 10 entries

id	Tenant_Id	owner
1	1	hpcms
2	2	guest

Showing 1 to 2 of 2 entries

Application管理及
可视化定义

新增资源分区

Show 10 entries

Search:

id	Namespace	Cpu Limits	Memory Limits	Service Limits	Pods Limit	PV Limit
2	dev env	10	1024	50	20	50
3	test env	5	1024	30	10	30
4	uat env	5	1024	30	8	30
5	default	20	1024	50	100	50

Showing 1 to 4 of 4 entries

新增资源分区

Namespace

Cpu Limits

Memory Limits

Pods Limit

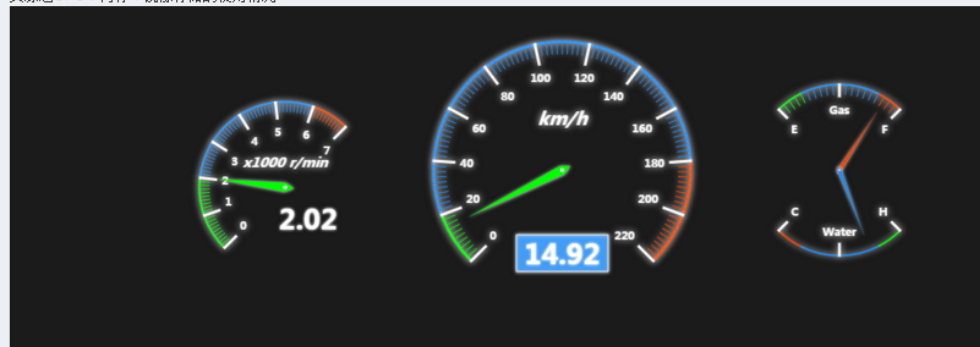
RC Limit

PV Limit

资源分区及配额管理

Ku8 eye开源项目

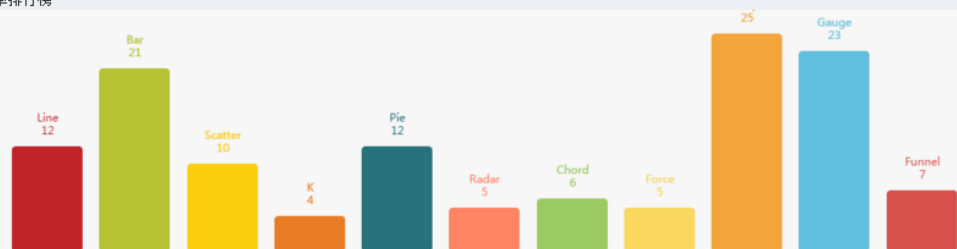
资源池CPU、内存、镜像存储的使用情况



CPU利用率排行榜



内存利用率排行榜



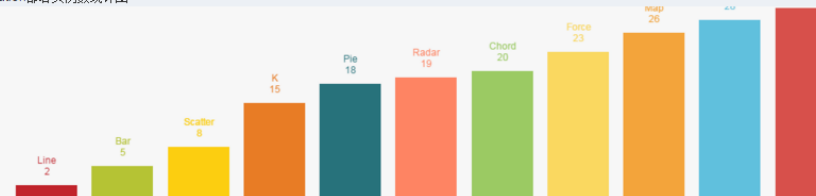
Application包含的Service个数统计图



Application包含的Pod个数统计图



Application部署实例数统计图



Thanks

FAQ时间