Визитка

СЕРГЕЙ ГОЛОВАШОВ,
ведущий инженер DevOps Bell Integrator

Визитка

НИКОЛАЙ СИТНИКОВ,
инженер DevOps Bell Integrator

Openshift и все вокруг него,
часть 3: начало работы, работа в CLI

В статье будет рассмотрено начало работы в качестве пользователя Openshift, работа в консольном клиенте, а также некоторые фичи консольного клиента (которые помогут в дальнейшей эксплуатации контейнеров) и темплеты (помогут деплоить на коластер сложно настраиваемые контейнеры, включающие дополнительные сущности — сервисы, роуты, конфиг мапы, секреты — с помощь одной команды и передачи переменных)

OpenShift состоит из двух типов медиан для создания и развертывания приложений с помощью графического интерфейса или интерфейса командной строки. В этой статье мы будем использовать CLI для создания нового приложения и клиент OC для связи со средой OpenShift.

Создание нового приложения

В OpenShift есть три способа создания нового приложения.

- Из исходного кода
- Из образа
- Из шаблона

Из исходного кода

Когда мы пытаемся создать приложение из исходного кода, OpenShift ищет файл Docker, который должен присутствовать в репозитории, и определяет процесс сборки приложения. Мы для создания приложения применим OC new-app.

Первое, что следует иметь в виду при использовании репозитория, это то, что он должен указывать на источник в репозитории, откуда OpenShift будет извлекать код и собирать его. Если репозиторий клонирован на компьютере Docker, на котором установлен клиент OC, и пользователь находится в том же каталоге, его можно создать с помощью следующей команды.

```
$ oc new-app .<текущая рабочая директория>
```

Ниже приведен пример попытки построить из удаленного репозитория для определенной ветви.

```
$ oc new-app https://github.com/openshift/Testing-deployment.
git#test1
```

Здесь test1 – это ветка, из которой мы пытаемся создать новое приложение в OpenShift.

При указании файла Docker в хранилище мы должны определить стратегию сборки, как показано ниже.

```
$ oc new-app OpenShift/OpenShift-test~https://github.com/
openshift/Testingdeployment.git
```

Из образа

При создании приложения с использованием образа, образ присутствует на локальном сервере Docker, во внутреннем хранилище Docker или в концентраторе Dock er. Единственное, в чем пользователь должен убедиться, это то, что он может без проблем извлекать образ из концентратора.

OpenShift имеет возможность определять используемый источник, будь то образ Docker или поток источника. Однако если пользователь желает, он может явно определить, является ли это потоком образа или образом Docker.

```
$ oc new-app - - docker-image tomcat
```

Использование потока образа –

```
$ oc new-app tomcat:v1
```

Из шаблона

Шаблоны могут быть использованы для создания нового приложения. Это может быть уже существующий шаблон или создание нового шаблона.

Следующий файл yaml – это шаблон, который можно использовать для развертывания.

Разработка и развертывание веб-приложения

Разработка нового приложения в OpenShift

Чтобы создать новое приложение в OpenShift, мы должны написать новый код приложения и собрать его, используя команды сборки OpenShift OC. Как уже говорилось, у нас есть несколько способов создания нового образа. Здесь

мы будем использовать шаблон для сборки приложения. Этот шаблон создаст новое приложение при запуске с помощью команды oc new-app.

Будет создан следующий шаблон: два интерфейсных приложения и одна база данных. Наряду с этим он создаст два новых сервиса, и эти приложения будут развернуты в кластере OpenShift. При создании и развертывании приложения сначала необходимо создать пространство имен в OpenShift и развернуть приложение в этом пространстве имен.

Создание нового пространства имен

```
$ oc new-project openshift-test --display-name = "OpenShift
Sample" -- description = "This is an example project to
demonstrate OpenShift"
```

Шаблон

```
{
   "kind": "Template",
   "apiVersion": "v1", "metadata": {
      "name": "openshift-helloworld-sample",
      "creationTimestamp": null,
         "annotations": {
         "description": "This example shows how to create
         a simple openshift application in openshift origin",
         "iconClass": "icon-openshift",
         "tags": "instant-app,openshift,mysql"
      }
   }
},
```

Определения объектов

Секретное определение в шаблоне

```
"objects": [
{
   "kind": "Secret",
   "apiVersion": "v1",
   "metadata": {"name": "dbsecret"},
   "stringData" : {
      "mysql-user" : "${MYSQL_USER}",
      "mysql-password" : "${MYSQL_PASSWORD}"
   }
},
```

Определение сервиса в шаблоне

```
{
   "kind": "Service",
   "apiVersion": "v1",
   "metadata": {
      "name": "frontend",
      "creationTimestamp": null
   },
   "spec": {
      "ports": [
         {
            "name": "web",
            "protocol": "TCP",
            "port": 5432,
            "targetPort": 8080,
            "nodePort": 0
         }
      ],
      "selector": {"name": "frontend"},
      "type": "ClusterIP",
      "sessionAffinity": "None"
   },
   "status": {
   "loadBalancer": {}
   }
},
```

Определение маршрута в шаблоне

```
{
   "kind": "Route",
   "apiVersion":
   "v1",
   "metadata": {
      "name": "route-edge",
      "creationTimestamp": null,
      "annotations": {
         "template.openshift.io/expose-uri":
         "http://{.spec.host}{.spec.path}"
      }
   },
   "spec": {
      "host": "www.example.com",
      "to": {
         "kind": "Service",
         "name": "frontend"
      },
      "tls": {
         "termination": "edge"
      }
   },
   "status": {}
},
{
   "kind": "ImageStream",
   "apiVersion": "v1",
   "metadata": {
      "name": "origin-openshift-sample",
      "creationTimestamp": null
   },
   "spec": {},
   "status": {
      "dockerImageRepository": ""
   }
},
{
   "kind": "ImageStream",
   "apiVersion": "v1",
   "metadata": {
      "name": "openshift-22-ubuntu7",
      "creationTimestamp": null
   },
   "spec": {
      "dockerImageRepository": "ubuntu/openshift-22-ubuntu7"
   },
   "status": { "dockerImageRepository": ""
   }
},
```

Создание определения конфигурации в шаблоне

```
{
   "kind":
   "BuildConfig",
   "apiVersion": "v1",
   "metadata":
   {
      "name": "openshift-sample-build",
      "creationTimestamp": null,
      "labels": {name": "openshift-sample-build"}
   },
   "spec": {
   "triggers": [
      { "type": "GitHub",
         "github": {
            "secret": "secret101" }
      },
      {
```

```
        "type": "Generic",
        "generic": {
           "secret": "secret101",
           "allowEnv": true }
      },
      {
         "type": "ImageChange",
         "imageChange": {}
      },
      { "type": "ConfigChange"}
   ],
   "source": {
      "type": "Git",
      "git": {
         "uri": https://github.com/openshift/
         openshift-hello-world.git }
   },
   "strategy": {
      "type": "Docker",
      "dockerStrategy":
         { "from": {
            "kind": "ImageStreamTag",
            "name": "openshift-22-ubuntu7:latest"
   },
   "env": [
               {
                  "name": "EXAMPLE",
                  "value": "sample-app"
               }
            ]
         }
      },
      "output": {
         "to": {
               "kind": "ImageStreamTag",
            "name": "origin-openshift-sample:latest"
         }
      },
      "postCommit": {
         "args": ["bundle", "exec", "rake", "test"]
      },
      "status": { "lastVersion": 0
      }
   }
},
```

Конфигурация развертывания в шаблоне

```
"status": {
   "lastVersion": 0
   }
{
   "kind": "DeploymentConfig",
   "apiVersion": "v1",
   "metadata": {
      "name": "frontend",
      "creationTimestamp": null
   }
},
"spec": {
   "strategy": {
      "type": "Rolling",
      "rollingParams": {
         "updatePeriodSeconds": 1,
         "intervalSeconds": 1,
         "timeoutSeconds": 120,
         "pre": {
            "failurePolicy": "Abort",
            "execNewPod": {
               "command": [
                  "/bin/true"
               ],
               "env": [
                  {
                     "name": "CUSTOM_VAR1",
                     "value": "custom_value1"
                  }
               ]
            }
         }
      }
   }
}
"triggers": [
   {
      "type": "ImageChange",
      "imageChangeParams": {
         "automatic": true,
         "containerNames": [
            "openshift-helloworld"
         ],
         "from": {
            "kind": "ImageStreamTag",
            "name": "origin-openshift-sample:latest"
         }
      }
   },
   {
      "type": "ConfigChange"
      }
   ],
   "replicas": 2,
   "selector": {
      "name": "frontend"
   },
   "template": {
   "metadata": {
      "creationTimestamp": null,
      "labels": {
         "name": "frontend"
      }
   },
   "spec": {
      "containers": [
      {
         "name": "openshift-helloworld",
         "image": "origin-openshift-sample",
         "ports": [
            {
            "containerPort": 8080,
            "protocol": "TCP"
            }
         ],
   "env": [
      {
         "name": "MYSQL_USER",
         "valueFrom": {
            "secretKeyRef" :
            {
               "name" : "dbsecret",
               "key" : "mysql-user"
            }
         }
      },
   {
      "name": "MYSQL_PASSWORD",
      "valueFrom": {
         secretKeyRef" :
         {
            "name" : "dbsecret",
            "key" : "mysql-password"
         }
      }
   },
   {
      "name": "MYSQL_DATABASE",
      "value": "${MYSQL_DATABASE}"
   }
],
   "resources": {},
   "terminationMessagePath": "/dev/termination-log",
   "imagePullPolicy": "IfNotPresent",
   "securityContext": {
      "capabilities": {},
      "privileged": false
      }
   }
],
```

```
      "restartPolicy": "Always",
      "dnsPolicy": "ClusterFirst"
   },
   "status": {}
},
```

Определение сервиса в шаблоне

```
{
   "kind": "Service",
   "apiVersion": "v1",
   "metadata": {
      "name": "database", "creationTimestamp": null
   },
   "spec": {
   "ports": [
      {
         "name": "db",
         "protocol": "TCP",
         "port": 5434,
         "targetPort": 3306,
         "nodePort": 0
      }
   ],
   "selector": {
      "name": "database
   },
   "type": "ClusterIP",
   "sessionAffinity": "None" },
   "status": {
      "loadBalancer": {}
   }
},
```

Определение конфигурации развертывания в шаблоне

```
{
   "kind": "DeploymentConfig",
   "apiVersion": "v1",
   "metadata": {
      "name": "database",
      "creationTimestamp": null
   },
   "spec": {
      "strategy": {
         "type": "Recreate",
         "resources": {}
      },
      "triggers": [
         {
            "type": "ConfigChange"
         }
      ],
      "replicas": 1,
      "selector": {"name": "database"},
      "template": {
         "metadata": {
            "creationTimestamp": null,
            "labels": {"name": "database"
         }
      },
      "template": {
         "metadata": {
            "creationTimestamp": null,
            "labels": {
               "name": "database"
            }
         },
         "spec": {
            "containers": [
               {
                  "name": "openshift-helloworld-database",
                  "image": "ubuntu/mysql-57-ubuntu7:latest",
                  "ports": [
                     {
                        "containerPort": 3306,
                        "protocol": "TCP"
                     }
                  ],
"env": [
   {
      "name": "MYSQL_USER",
      "valueFrom": {
         "secretKeyRef" : {
            "name" : "dbsecret",
            "key" : "mysql-user"
         }
      }
   },
{
   "name": "MYSQL_PASSWORD",
   "valueFrom": {
      secretKeyRef" : {
         "name" : "dbsecret",
         "key" : "mysql-password"
      }
   }
},

{
   "name": "MYSQL_DATABASE",
    "value": "${MYSQL_DATABASE}"
   }
],
   "resources": {},
   "volumeMounts": [
      {
         "name": "openshift-helloworld-data",
         "mountPath": "/var/lib/mysql/data"
      }
   ],
   "terminationMessagePath": "/dev/termination-log",
   "imagePullPolicy": "Always",
   "securityContext": {
      "capabilities": {},
      "privileged": false
      }
   }
],
      "volumes": [
         {
            "name": "openshift-helloworld-data",
            "emptyDir": {"medium":
             ""}
         }
      ],
         "restartPolicy": "Always",
         "dnsPolicy": "ClusterFirst"
         }
      }
   },
   "status": {}
},
"parameters": [
   {
      "name": "MYSQL_USER",
      "description": "database username",
      "generate": "expression",
      "from": "user[A-Z0-9]{3}",
      "required": true
   },
{
   "name": "MYSQL_PASSWORD",
   "description": "database password",
   "generate": "expression",
   "from": "[a-zA-Z0-9]{8}",
   "required": true
},
{
   "name": "MYSQL_DATABASE",
   "description": "database name",
   "value": "root",
   "required": true
   }
],
   "labels": {
      "template": "application-template-dockerbuild"
   }
}
```

Приведенный выше файл шаблона необходимо скомпилировать сразу. Нам нужно сначала скопировать весь контент в один файл и назвать его как файл yaml, после того как он будет сделан.

Чтобы создать приложение, нужно выполнить следующую команду:

```
$ oc new-app application-template-stibuild.json
--> Deploying template openshift-helloworld-sample for
"application-template-stibuild. json"

   openshift-helloworld-sample
   -----
   This example shows how to create a simple ruby application
in openshift origin v3
   *  With parameters:
      *  MYSQL_USER = userPJJ # generated
      *  MYSQL_PASSWORD = cJHNK3se # generated
      *  MYSQL_DATABASE = root

--> Creating resources with label app = ruby-helloworld-sample ...
   service "frontend" created
   route "route-edge" created
   imagestream "origin-ruby-sample" created
   imagestream "ruby-22-centos7" created
   buildconfig "ruby-sample-build" created
   deploymentconfig "frontend" created
   service "database" created
   deploymentconfig "database" created

--> Success
   Build scheduled, use 'oc logs -f bc/ruby-sample-build'
   to track its progress. Run 'oc status' to view your app.
```

Если мы хотим отслеживать сборку, это можно сделать с помощью

```
$ oc get builds
NAME    TYPE              FROM                STATUS     STARTED     DURATION
openshift-sample-build-1  Source  Git@bd94cbb    Running    7 seconds ago  7s
```

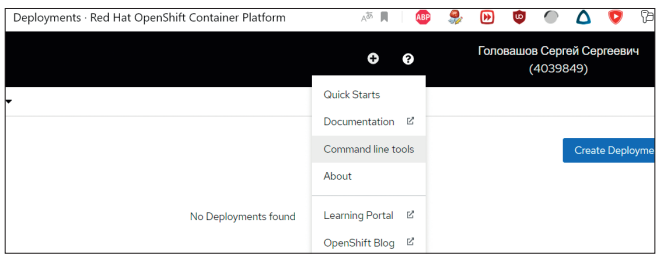
Мы можем проверить развернутые приложения в OpenShift, используя

```
$ oc get pods
NAME                          READY    STATUS    RESTARTS AGE
database-1-le4wx                1/1    Running     0      1m
frontend-1-e572n                1/1    Running     0      27s
frontend-1-votq4                1/1    Running     0      31s
opeshift-sample-build-1-build   0/1    Completed   0      1m
```

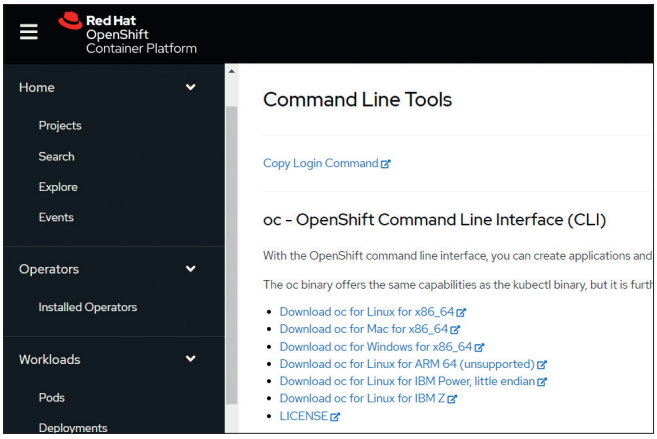
Мы можем проверить, созданы ли сервисы приложений согласно определению сервиса.

Работа с Openshift CLI

Самый просто способ получить консольного клиента Openshift – это скачать его с помощью консоли. Нажимаем на «+», выбираем Commnd line tool:



Скачиваем нужную нам версию:



OpenShift CLI способен выполнять все основные и расширенные настройки управления, добавления и развертывания приложений. Это крутая команда, она здорово продумана, она мощная, гибкая и у нее, как вы увидите, есть много скрытых возможностей, которые стоит попробовать.

Начнем с базовых вещей:

> Войти в OpenShift контейнер Platform сервера:

```
oc login
```

> Создать новый проект:

```
oc new-project {{project_name}}
```

> Перейти в существующий проект:

```
oc project {{project_name}}
```

> Добавить новое приложение в проект:

```
oc new-app {{repo_url}} --name {{application}}
```

> Открыть удалённый консольный интерфейс в контейнере:

```
oc rsh {{pod_name}}
```

> Вывести список задач проекта:

```
oc get pods
```

> Выйти из текущей сессии:

```
oc logout
```

А теперь переходим к фичам, которые могут пригодиться.

1. Сначала отладка

Когда я не знаю, что происходит, или получаю непонятное сообщение об ошибке, я всегда использую флаг **--loglevell**, который включает запись лога в stderr. В зависимости от значения этого флага можно увидеть curl-вызовы API Rest, содержание ответов API Rest или даже более детализированную информацию.

--loglevell	Output
1-5	Normal
6	API request

--loglevell	Output
7	API request API request headers
8	API request API request headers [API request body] API response headers API response body
9-10	API request API request headers [API request body] API response headers Curl request

```
$ oc --loglevel 7 get pod
...
I0216 21:24:12.027793 973 cached_discovery.go:72]
returning cached discovery info from /home/jtudelag/.
kube/192.168.42.77_8443/v1/serverresources.json
I0216 21:24:12.028046 973 round_trippers.go:383] GET
https://192.168.42.77:8443/api/v1/namespaces/myproject/pods
I0216 21:24:12.028052 973 round_trippers.go:390] Request
Headers:
I0216 21:24:12.028057 973 round_trippers.go:393] Accept:
application/json
I0216 21:24:12.028061 973 round_trippers.go:393] User-Agent:
oc/v1.7.6+a08f5eeb62 (linux/amd64) kubernetes/c84beff
I0216 21:24:12.053230 973 round_trippers.go:408] Response
Status: 200 OK in 25 milliseconds
I0216 21:24:12.055143 973 cached_discovery.go:119]
returning cached discovery info from /home/jtudelag/.
kube/192.168.42.77_8443/servergroups.json
I0216 21:24:12.055228 973 cached_discovery.go:72]
returning cached discovery info from /home/jtudelag/.
kube/192.168.42.77_8443/authentication.k8s.io/v1/
serverresources.json
I0216 21:24:12.055288 973 cached_discovery.go:72]
...
```

Например, loglevel 9 очень удобен, когда вы патчите OCP-объект, поскольку позволяет увидеть сам патч (содержание API-запроса).

Если, допустим, патч меняет метку сервиса на «app: hello-jorge», то это будет выглядеть так:

```
$ oc --loglevel 9 edit svc hello-openshift
...
I0216 21:33:15.786463 1389 request.go:994] Request Body:
{"metadata":{"labels":{"app":"hello-jorge"}}}
I0216 21:33:15.786590 1389 round_trippers.go:386] curl -k
-v -XPATCH -H "Accept: application/json" -H "Content-Type:
application/strategic-merge-patch+json" -H "User-Agent:
oc/v1.7.6+a08f5eeb62 (linux/amd64) kubernetes/c84beff"
https://192.168.42.77:8443/api/v1/namespaces/myproject/
services/hello-openshift
I0216 21:33:15.797185 1389 round_trippers.go:405] PATCH
https://192.168.42.77:8443/api/v1/namespaces/myproject/
services/hello-openshift 200 OK in 10 milliseconds
...
```

Примечание. В моменты отчаяния вместо одной девятки можно вбивать сразу несколько. Вывод команды OC от этого не изменится, но, возможно, вам полегчает.

```
$ oc --loglevel 9999 get pod
```

2. su –

Да, вы правильно поняли. Команду OC можно запустить от имени другого пользователя, или, говоря на языке OCP, использовать имперсонацию (https://docs.openshift.com/container-platform/3.7/architecture/additional_concepts/authentication.html#authentication-impersonation). Разумеется, при наличии соответствующих прав. И для этого достаточно всего лишь использовать флаг **--as**.

Например:

```
# запускаем от имени пользователя jorge
$ oc --as=jorge get pods
```

Имперсонация работает не только для пользователей, но и для групп:

```
# запускаем от имени группы developers
$ oc --as-group=developers get pods
```

Имперсонация пригодится в самых разных случаях. Например, когда надо проверить, сможет ли пользователь выполнить то или иное действие, или посмотреть, что ему выдаст команда OC. Еще имперсонация очень помогает при неразберихе с ролями и разрешениями.

Что делать, если у вас есть токен, но вы не его владелец? В этом случае можно войти в OpenShift с помощью этого токена, а затем выполнить команду OC whoami

3. Whoami

Команда OC whoami знакома, наверное, всем. Особенно флаг -t, позволяющий получить токен носителя для текущего пользователя/сеанса. Но что делать, если у вас есть токен, но вы не его владелец? В этом случае можно войти в OpenShift с помощью этого токена, а затем выполнить команду OC whoami. Хотя, подождите, можно же сразу узнать имя владельца, просто передав токен команде OC whoami третьим аргументом без всяких флагов.

Смотрите:

```
# сохраняем токен
$ token=$(oc whoami -t)

# получаем имя владельца токена
$ oc whoami $token
jorge
```

4. OC debug

Как известно, shell можно запустить прямо в работающем pod'e. Иногда бывает полезно сделать полную копию конфигурации запущенного pod'a и устранять неполадки через shell. Это так называемый метод по умолчанию. А теперь взгляните, что позволяют сделать опции OC debug: можно запустить контейнер от имени root или любого другого пользователя; можно запустить его на выбранном узле или можно запустить в нем не shell, а другую команду.

При этом надо указывать верный dc, например:

```
# получаем shell внутри pod'а dc/jorge
$ oc debug dc/jorge

# то же самое, но как root
$ oc debug --as-root=true dc/jorge
```

5. OC explain

В объектах OpenShift/Kubernetes иногда бывает огромное множество полей. В поисках примеров определений таких объектов часто приходится обращаться к документации по OCP или другим первоисточникам. Однако с тем же успехом можно использовать команду **OC explain**.

Эта команда выводит документацию по ресурсам и их полям, что бывает очень полезно при декларировании новых объектов OCP или в тех случаях, когда у вас нет доступа к официальной документации OCP.

Например, вот как можно получить документацию по pod'ам и описание affinity-полей:

```
# получаем справку по pod'у
$ oc explain pod
DESCRIPTION:
Pod is a collection of containers that can run on a host.
This resource is created by clients and scheduled onto hosts.

FIELDS:
   metadata <Object>
      Standard object's metadata. More info:
      http://releases.k8s.io/HEAD/docs/devel/api-conventions.
md#metadata

   spec <Object>
      Specification of the desired behavior of the pod.
More info:
      http://releases.k8s.io/HEAD/docs/devel/api-conventions.
md#spec-and-status

   status <Object>
      Most recently observed status of the pod. This data may
not be up to date.
      Populated by the system. Read-only. More info:
      http://releases.k8s.io/HEAD/docs/devel/api-conventions.
md#spec-and-status

   apiVersion  <string>
      APIVersion defines the versioned schema of this
representation of an
      object. Servers should convert recognized schemas to
the latest internal
      value, and may reject unrecognized values. More info:
      http://releases.k8s.io/HEAD/docs/devel/api-conventions.
md#resources

   kind <string>
      Kind is a string value representing the REST resource
this object
      represents. Servers may infer this from the endpoint
the client submits
      requests to. Cannot be updated. In CamelCase. More
info:
      http://releases.k8s.io/HEAD/docs/devel/api-conventions.
md#types-kinds

# получаем описание полей affinity
$ oc explain pod.spec.affinity
RESOURCE: affinity <Object>

DESCRIPTION:
      If specified, the pod's scheduling constraints

    Affinity is a group of affinity scheduling rules.
```

В объектах OpenShift/Kubernetes иногда бывает огромное множество полей. В поисках примеров определений таких объектов часто приходится обращаться к документации по OCP

```
FIELDS:
   nodeAffinity <Object>
      Describes node affinity scheduling rules for the pod.

   podAffinity <Object>
      Describes pod affinity scheduling rules (e.g. co-locate
this pod in the
      same node, zone, etc. as some other pod(s)).

   podAntiAffinity <Object>
      Describes pod anti-affinity scheduling rules
(e.g. avoid putting this pod
      in the same node, zone, etc. as some other pod(s)).
```

6. Забудьте про grep, awk, cut и т. п.

Еще одна крутая фишка команды OC – это встроенные функции форматирования вывода. Про опции **-o json** или **-o yaml** знают все, но у флага -o есть и множество других опций.

Самые мощные, возможно, – это **go-template** и **jsonpath**:

```
json|yaml|wide|name|custom-columns=...
|custom-columns-file=...|go-template=...
|go-template-file=...|jsonpath=...|jsonpath-file=...
```

Скажем, вы хотите узнать, какой сервис предоставляется определенным маршрутом (маршрутом docker-реестра):

```
# запросим сервисы, предоставляемые маршрутами, но только
для узла с именем my-docker-registry.example.com
$ oc get routes -o=go-template='{{range .items}}{{if eq
.spec.host "my-docker-registry.example.com"}}{{.metadata.
name}}{{end}}{{end}}'
docker-registry
```

Или, допустим, нужно узнать стратегию развертывания маршрутизатора router dc:

```
# запросим стратегию развертывания маршрутизатора
$ oc get dc router -o=go-template='{{ .spec.strategy.type }}'
Rolling
```

Как видите, OC – это удивительная команда. С ней определенно стоит поиграться, поскольку это одна из самых крутых вещей в OpenShift.

• • •

На этом заканчиваем третью часть нашего цикла про Openshift. В следующей статье мы продолжим рассказывать про администрирование кластера и управления им. EOF

Ключевые слова: управление Openshift, консольный клиент, работа с контейнерами, шаблоны