Вакансия: AI-разработчик

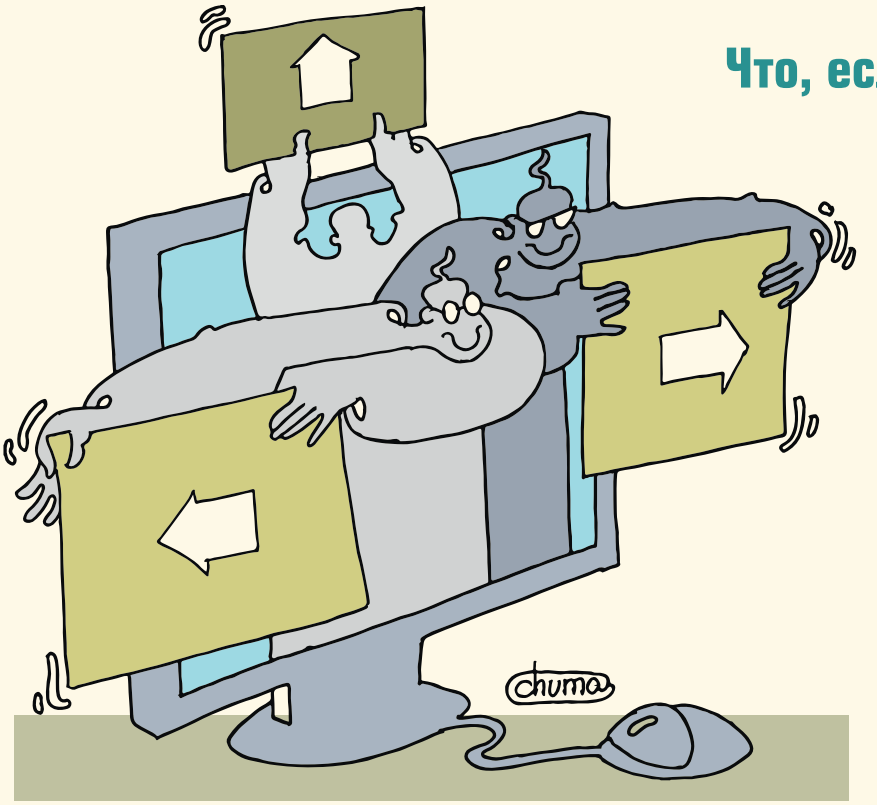
ISSN 1813-5579
22006
9 771813 557005

Системный администратор

ежемесячный журнал www.samag.ru

№6(235)
2022

Нам бы день продержаться…



HP Integrity VM – виртуализируй и властвуй

Что, если мы были взломаны? Мониторинг угроз с помощью исторического анализа в TI-платформах

Openshift и все вокруг него

Как российские вузы и компании переходят на свободное ПО

СЛЕТ ДСА-2022

Не хочу лениться! Хочу учиться?

Наука и технологии

Наука и технологии

16+

Разработка методов и средств контроля пространственных рисков на базе банков данных и интегрированных цифровых карт

Использование муравьиного алгоритма в задачах построения расписаний конвейеризированных проектов

Визитка

СЕРГЕЙ ГОЛОВАШОВ,
ведущий инженер DevOps, руководитель центра компетенций, компания Bell Integrator

Визитка

НИКОЛАЙ СИТНИКОВ,
инженер DevOps, компания Bell Integrator

Openshift и все вокруг него

Часть 5: Масштабирование приложений, стратегии деплоя, администрирование

Продолжение серии статей про Openshift, его администрирование и работу в нем. В данной статье будет рассмотрено управление кластером Openshift, добавление и удаление нод в состав кластера, методы управления узлами, логирование и журналирование, а также обновление узлов.

МАСШТАБИРОВАНИЕ ПРИЛОЖЕНИЯ

Автоматическое масштабирование – это функция в OpenShift, где развертываемые приложения могут масштабироваться и уменьшаться в зависимости от требований в соответствии с определенными спецификациями. Существует два типа масштабирования приложения.

Вертикальное масштабирование (VPA)

Вертикальное масштабирование – это все больше и больше энергии на одной машине, что означает добавление процессора и жесткого диска. Это старый метод OpenShift, который сейчас не поддерживается в выпусках. На самом деле он продолжает работать, но не подчиняется стандартной системе настройки.Метод получил свое развитие в виде выделения ресурсов поде и определения их в разделе resources. Существуют два метода выделения ресурсов поде – qos guarantee и qos Burstable.

> Вертикальное масштабирование — это все больше и больше энергии на одной машине, что означает добавление процессора и жесткого диска. Это старый метод OpenShift, который сейчас не поддерживается в выпусках

QoS (качество обслуживания) – в основном переводится как «Уровень качества обслуживания» или как «Гарантия качества обслуживания». Представляет собой конфигурацию, которая действует на под. Когда Kubernetes создает под в Openshift, он назначает ему уровень QoS, который может быть одним из следующих:

- **Guaranteed:** каждый контейнер в модуле должен иметь ограничения и запросы памяти/ЦП, и значения должны быть одинаковыми. Если контейнер указывает только ограничение, не задавая запроса, значение запроса будет равно значению лимита.
- **Burstable:** по крайней мере, один контейнер в модуле имеет запрос памяти или ЦП и не соответствует требованиям уровня гарантии, то есть настройки значений памяти/ЦП отличаются.
- **BestEffort:** контейнер не должен иметь ограничений или запросов памяти или ЦП.

Конфигурация настраивается не с помощью элемента конфигурации, а путем настройки CPU/MEM.limits, либо requests, тем самым устанавливая числовое ограничение на использование ресурсов, и, как следствие, уровень качества обслуживания.

Ниже пример Guaranteed:

```
spec:
   containers:
   - name: demo
      image: mritd/demo
      resources:
         limits:
            cpu: 100m
            memory: 100Mi
         requests:
            cpu: 100m
            memory: 100Mi
      ports:
      - containerPort: 80
```

Ниже пример Burstable:

```
spec:
   containers:
   - name: demo
      image: mritd/demo
      resources:
```

```
            limits:
               cpu: 100m
               memory: 100Mi
            requests:
               cpu: 10m
               memory: 10Mi
         ports:
         - containerPort: 80
```

Ниже пример BestEffort:

```
   spec:
      containers:
      - name: demo
         image: mritd/demo
         ports:
         - containerPort: 80
```

Ограничения ресурсов и запросы требований к ресурсам

Для CPU: если ЦП, используемый службой в поде, превышает установленные ограничения, модуль не будет уничтожен, но будет ограничен. Если ограничения не установлены, модуль может использовать все свободные ресурсы процессора.

На память: когда модуль использует больше памяти, чем установленный предел, модуль container Процесс будет kernel, так как OOM kill Падение. Когда контейнер прекращается из-за OOM, система имеет тенденцию перезапускать контейнер или воссоздавать модуль на исходной машине.

Ниже приведены три способа ограничения ресурсов с разными областями действия.

> **На Node** - резервирование ресурсов узла в kubelet, конфигурация с помощью командной строки (реализуемо не во всех версиях утилиты ос)

Следующая конфигурация настраивается в соответствии с реальной ситуацией.

```
--system-reserved=cpu=500m,memory=1.5Gi \
--eviction-hard=memory.available<1.5Gi,nodefs.available<20Gi,
imagefs.available<15Gi \
--eviction-soft=memory.available<2Gi,nodefs.available<25Gi,
imagefs.available<10Gi \
--eviction-soft-grace-period=memory.available=2m,
nodefs.available=2m,imagefs.available=2m \
--eviction-max-pod-grace-period=30 \
--eviction-minimum-reclaim=memory.available=200Mi,
nodefs.available=5Gi,imagefs.available=5Gi
```

> На **namespace** конфигурации ограничения

```
apiVersion: "v1"
kind: "LimitRange"
metadata:
   name: "test-resource-limits"
   namespace: test
spec:
   limits:
      - type: "Pod"
         max:
            cpu: "2"
            memory: "2Gi"
         min:
            cpu: "30m"
            memory: "50Mi"
      - type: "Container"
         max:
            cpu: "2"
            memory: "2Gi"
         min:
            cpu: "30m"
            memory: "50Mi"
         default:
            cpu: "300m"
            memory: "600Mi"
         defaultRequest:
            cpu: "30m"
            memory: "50Mi"
         maxLimitRequestRatio:
            cpu: "30"
```

> На **Pod** ограничения можно использовать Deployment

```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
   name: demo-deployment
spec:
   replicas: 1
   template:
      metadata:
         annotations:
            sidecar.istio.io/inject: "false"
         labels:
            app: demo
      spec:
         containers:
         - name: demo
            image: mritd/demo
            resources:
               limits:
                  cpu: 100m
                  memory: 100Mi
               requests:
                  cpu: 10m
                  memory: 10Mi
            ports:
            - containerPort: 80
```

QoS приоритет

Три уровня приоритета QoS, от высокого до низкого (слева направо): **Guaranteed** → **Burstable** → **BestEffort**.

Стратегия утилизации ресурсов OpenShift и его оркестратора на базе Kubernetes

Стратегия утилизации ресурсов: когда кластер контролирует node, когда ресурсы памяти или ЦП узла исчерпаны, для защиты нормальной работы узла будет активирована стратегия восстановления ресурсов, чтобы уменьшить занятость ресурсов путем изгнания Pod на узле.

Три политики QoS исключены по приоритету, от высокого к низкому (слева направо): **BestEffort** → **Burstable** → **Guaranteed**.

Рекомендации:

> Если ресурсов достаточно, вы можете установить QoS пода на Гарантированный

> QoS модуля менее важной службы устанавливается на Burstable или BestEffort. Такие как filebeat, logstash, fluentd и т.д.

Предложение: при установке k8s рекомендуется использовать swap раздел. Именно он может решить проблему нехватки памяти, когда её, памяти, недостаточно. Используйте Swap, когда нагрузка на систему слишком высока или оркестратор будет занимать все время диска на операции чтения/записи (disk io).

Горизонтальное масштабирование (HPA)

Этот тип масштабирования полезен, когда необходимо обработать больше запросов, увеличив число машин.

Он не исключает вертикальное масштабирование, но очень хорошо расширяет его возможности по поднятию необходимого дополнительного количества реплик приложения, когда ресурсы одной поды уже исчерпаны.

В OpenShift есть два способа включить функцию масштабирования.

> Использование файла конфигурации развертывания
> Во время запуска образов

Использование файла конфигурации развертывания

В этом методе функция масштабирования включается через файл yaml конфигурации deploymant. Для этого используется команда OC autoscale с минимальным и максимальным количеством реплик, которые необходимо запустить в любой момент времени в кластере. Нам нужно определение объекта для создания автомасштабатора. Ниже приведен пример файла определения pod autoscaler:

```
apiVersion:
extensions/v1beta1 kind:
HorizontalPodAutoscaler metadata:
   name:
   database spec:
      scaleRef:
         kind:
         DeploymentConfig
         name: database
         apiVersion: v1
         subresource: scale
      minReplicas: 1
      maxReplicas: 10
      cpuUtilization:
         targetPercentage: 80
```

Как только у нас будет файл, нам нужно сохранить его в формате yaml и выполнить следующую команду для развертывания:

```
$ oc create -f <file name>.yaml
```

Во время запуска образа

Можно также автоматически масштабировать без файла yaml, используя следующую команду oc autoscale в командной строке oc:

```
$ oc autoscale dc/database --min 1 --max 5 --cpu-percent = 75
deploymentconfig "database" autoscaled
```

Эта команда также создаст файл аналогичного типа, который впоследствии можно будет использовать для справки.

СТРАТЕГИИ ДЕПЛОЯ

Стратегия развертывания в OpenShift определяет поток развертывания с различными доступными методами. В OpenShift ниже перечислены важные типы стратегий развертывания.

> Роллинг стратегия
> Воссоздать стратегию
> Индивидуальная стратегия

Rolling стратегия

Ниже приведен пример файла конфигурации развертывания, который используется в основном для развертывания на узлах OpenShift:

```
   kind:
   "DeploymentConfig"
   apiVersion: "v1"
   metadata:
      name: "database" spec:
         template:
   metadata:
   labels:
      name: "Database1"
   spec:
      containers:
      -  name: "vipinopenshifttest"
         image: "openshift/mongoDB"
         ports:
            -  containerPort: 8080
               protocol: "TCP"
   replicas: 5
   selector:
      name: "database"
      triggers:
-  type: "ConfigChange"
   -  type: "ImageChange"
      imageChangeParams:
         automatic: true
         containerNames:
         -  "vipinopenshifttest"
         from:
            kind: "ImageStreamTag"
            name: "mongoDB:latest"
         strategy:
          type: "Rolling
```

Скользящая стратегия

Скользящая стратегия используется для непрерывного обновления или развертывания. Этот процесс также поддерживает куки жизненного цикла, которые используются для внедрения кода в любой процесс развертывания.

Воссоздать стратегию

Эта стратегия развертывания имеет некоторые основные функции стратегии развертывания и поддерживает ловушку жизненного цикла.

Индивидуальная стратегия

Это очень полезно, когда кто-то хочет предоставить свой собственный процесс или процесс развертывания. Все настройки могут быть сделаны согласно требованию:

```
strategy:
   type: Recreate
   recreateParams:
      pre: {}
      mid: {}
      post: {}
   strategy:
      type: Rolling rollingParams:
         timeoutSeconds: <time in seconds>
         maxSurge: "<definition in %>"
         maxUnavailable: "<Defintion in %>" pre: {}
post: {}

$ oc deploy <deployment_config> --latest
```

АДМИНИСТРИРОВАНИЕ

Конфигурация мастера и узла

В OpenShift нам нужно использовать команду start вместе с OC для загрузки нового сервера. При запуске нового мастера нам нужно использовать мастер вместе с командой

запуска, тогда как при запуске нового узла нам нужно использовать узел вместе с командой запуска. Чтобы сделать это, нам нужно создать файлы конфигурации для мастера, а также для узлов. Мы можем создать базовый файл конфигурации для мастера и узла, используя следующую команду:

Для файла конфигурации узла:

```
$ oadm create-node-config --node-dir = /openshift.local.
config/node-<node_hostname>
```

Выполнив следующие команды, мы получим базовые файлы конфигурации, которые можно использовать в качестве отправной точки для конфигурации. Позже мы можем иметь тот же файл для загрузки новых серверов.

```
apiLevels:
   -  v1beta3
   -  v1
      apiVersion: v1
      assetConfig:
         logoutURL: ""
         masterPublicURL:
         https://172.10.12.1:7449 publicURL:
         https://172.10.2.2:7449/console/
         servingInfo:
            bindAddress:
            0.0.0.0:7449 certFile:
            master.server.crt clientCA:
               ""
      keyFile:
         master.server.key
         maxRequestsInFlight: 0
         requestTimeoutSeconds: 0
         controllers: '*'
         corsAllowedOrigins:
         - 172.10.2.2:7449
         - 127.0.0.1
         - localhost

$ openshift start master --write-config =
openshift.local.config/master

            dnsConfig:
               bindAddress: 0.0.0.0:53
            etcdClientInfo:
               ca: ca.crt
               certFile: master.etcd-client.crt
               keyFile: master.etcd-client.key
               urls:
               - https://10.0.2.15:4001
            etcdConfig:
               address: 10.0.2.15:4001
               peerAddress: 10.0.2.15:7001
               peerServingInfo:
                  bindAddress: 0.0.0.0:7001
                  certFile: etcd.server.crt
                  clientCA: ca.crt
                  keyFile: etcd.server.key
               servingInfo:
                  bindAddress: 0.0.0.0:4001
                  certFile: etcd.server.crt
                  clientCA: ca.crt
                  keyFile: etcd.server.key
            storageDirectory: /root/openshift.local.etcd
               etcdStorageConfig:
                  kubernetesStoragePrefix: kubernetes.io
                  kubernetesStorageVersion: v1
                  openShiftStoragePrefix: openshift.io
                  openShiftStorageVersion: v1
               imageConfig:
                  format:
                  openshift/origin${component}:${version}
                  latest: false
               kind: MasterConfig
               kubeletClientInfo:
                     ca: ca.crt
                     certFile: master.kubelet-client.crt
                     keyFile: master.kubelet-client.key
                     port: 10250
                     kubernetesMasterConfig:
                        apiLevels:
                           -  v1beta3
                           -  v1
                        apiServerArguments: null
                        controllerArguments: null
                     masterCount: 1
                   masterIP: 10.0.2.15
                     podEvictionTimeout: 5m
                     schedulerConfigFile: ""
                     servicesNodePortRange: 30000-32767
                     servicesSubnet: 172.30.0.0/16
                     staticNodeNames: []
                  masterClients: externalKubernetesKube
                  Config: ""
                     openshiftLoopbackKubeConfig:
                     openshift-master.kubeconfig masterPublic
                  URL: https://172.10.2.2:7449
               networkConfig:
                  clusterNetworkCIDR: 10.1.0.0/16
                  hostSubnetLength: 8
                  networkPluginName: ""
                  serviceNetworkCIDR: 172.30.0.0/16
               oauthConfig:
                  assetPublicURL:
                  https://172.10.2.2:7449/console/grantConfig:
                     method: auto
                     identityProviders:
                        -  challenge:
                        -  true login: true
                     name: anypasswordprovider:
                        apiVersion: v1
                        kind: AllowAllPasswordIdentityProvider
                     masterPublicURL: https://172.10.2.2:7449/
                     masterURL: https://172.10.2.2:7449/
                     sessionConfig:
                        sessionMaxAgeSeconds: 300
                        sessionName: ssn
                        sessionSecretsFile: ""
                     tokenConfig:
                        accessTokenMaxAgeSeconds: 86400
                        authorizeTokenMaxAgeSeconds: 300
               policyConfig:
                  bootstrapPolicyFile: policy.json
                  openshiftInfrastructureNamespace: openshift-infra
                  openshiftSharedResourcesNamespace: openshift
               projectConfig:
                  defaultNodeSelector: ""
                  projectRequestMessage: ""
                  projectRequestTemplate: ""
                  securityAllocator:
                     mcsAllocatorRange: s0:/2
                     mcsLabelsPerProject: 5
               uidAllocatorRange: 1000000000-1999999999/10000
               routingConfig:
                  subdomain: router.default.svc.cluster.local
               serviceAccountConfig:
               managedNames:
                  -  default
                  -  builder
                  - deployer
               -  masterCA: ca.crt
               -  privateKeyFile: serviceaccounts.private.key
                  privateKeyFile: serviceaccounts.private.key
                  publicKeyFiles:
                  - serviceaccounts.public.key
                  servingInfo:
                  -  bindAddress: 0.0.0.0:8443
                     certFile: master.server.crt
                     clientCA: ca.crt
                  -  keyFile: master.server.key
                     maxRequestsInFlight: 0
                  -  requestTimeoutSeconds: 3600
```

Файлы конфигурации узла

Вот так выглядят файлы конфигурации узла. Как только у нас есть эти файлы конфигурации, мы можем запустить следующую команду для создания главного сервера и сервера узлов.

```
$ openshift start --master-config = /openshift.local.config/
master/master- config.yaml --node-config = /openshift.local.
config/node-<node_hostname>/node- config.yaml

allowDisabledDocker:
   true apiVersion: v1
   dnsDomain: cluster.local
   dnsIP: 172.10.2.2
   dockerConfig:
      execHandlerName:
      native
   imageConfig:
      format: openshift/origin-${component}:${version}
      latest: false
   kind: NodeConfig
   masterKubeConfig: node.kubeconfig
   networkConfig:
      mtu: 1450
      networkPluginName: ""
   nodeIP: ""
   nodeName: node1.example.com

   podManifestConfig:
      path: "/path/to/pod-manifest-file" file
      CheckIntervalSeconds: 30
   servingInfo:
      bindAddress: 0.0.0.0:10250
      certFile: server.crt
      clientCA: node-client-ca.crt
      keyFile: server.key
   volumeDirectory: /root/openshift.local.volumes
```

Управляющие узлы

В OpenShift у нас есть утилита командной строки OC, которая в основном используется для выполнения всех операций в OpenShift. Мы можем использовать следующие команды для управления узлами:

> Для перечисления узла

```
$ oc get nodes
```

> Описание деталей об узле

```
$ oc describe node <node name>
```

> Удаление узла

```
$ oc delete node <node name>
```

> Список модулей на узле

```
$ oadm manage-node <node1> <node2> --list-pods [--pod-
selector=<pod_selector>] [-o json|yaml]
```

> Оценка стручков на узле

```
$ oadm manage-node <node1> <node2> --evacuate --dry-run
[--pod-selector=<pod_selec- tor>]
```

В мастере OpenShift есть встроенный сервер OAuth, который можно использовать для управления аутентификацией. Все пользователи OpenShift получают токен с этого сервера, который помогает им общаться с OpenShift API.

В OpenShift есть разные уровни аутентификации, которые можно настроить вместе с основным файлом конфигурации.

> Позволять все
> Запретить все
> Htpasswd
> LDAP
> Базовая аутентификация
> Заголовок запроса

Определяя основную конфигурацию, мы можем определить политику идентификации, где мы можем определить тип политики, которую мы хотим использовать.

Позволять все

```
oauthConfig:
   ...
   identityProviders:
   - name:
      Allow_Authontication
      challenge: true
      login: true
      provider:
         apiVersion: v1
         kind: AllowAllPasswordIdentityProvider
```

Запретить все

Это запретит доступ ко всем именам пользователей и паролям

```
oauthConfig:
   ...
   identityProviders:
   -  name:
         deny_Authontication
         challenge: true
         login: true
         provider:
            apiVersion: v1
            kind: DenyAllPasswordIdentityProvider
```

Htpasswd

HTPasswd используется для проверки имени пользователя и пароля по отношению к зашифрованному файлу пароля. Для создания зашифрованного файла есть следующая команда:

Использование зашифрованного файла.

```
oauthConfig:
   ...
   identityProviders:
   - name:
      htpasswd_authontication
      challenge: true
      login: true provider:
         apiVersion: v1
         kind: HTPasswdPasswordIdentityProvider
         file: /path/to/users.htpasswd

oauthConfig:
   ...
   identityProviders:
   - name:
      "ldap_authontication" challenge: true
      login: true
      provider:
         apiVersion: v1
         kind: LDAPPasswordIdentityProvider
         attributes:
            id:
```

```
            - dn
            email:
            - mail
            name:
            - cn preferredUsername:
               - uid
         bindDN: ""
         bindPassword: ""
         ca: my-ldap-ca-bundle.crt insecure: false
      url: "ldap://ldap.example.com/ou=users,dc=acme,dc=com?uid"


$ htpasswd </path/to/users.htpasswd> <user_name>
```

Базовая аутентификация

Это используется, когда проверка имени пользователя и пароля выполняется на основе межсерверной аутентификации. Аутентификация защищена в базовом URL и представлена в формате JSON.

Настройка учетной записи службы

Учетные записи служб предоставляют гибкий способ доступа к API OpenShift, предоставляя имя пользователя и пароль для аутентификации.

Включение учетной записи службы

Сервисная учетная запись использует для аутентификации пару ключей – открытый и закрытый. Аутентификация в API выполняется с использованием закрытого ключа и проверки его по открытому ключу.

```
   ServiceAccountConfig:
      ...
      masterCA: ca.crt
      privateKeyFile: serviceaccounts.private.key publicKeyFiles:
         - serviceaccounts.public.key
         - ...

$ Openshift cli create service account <name of server account>
```

Создание учетной записи службы

Используйте следующую команду для создания учетной записи службы:

```
oauthConfig:
   ...
   identityProviders:
   - name:
      my_remote_basic_auth_provider
      challenge: true
      login: true provider:
         apiVersion: v1
         kind: BasicAuthPasswordIdentityProvider
         url: https://www.vklnld908.int.example.com/
         remote-idp ca: /path/to/ca.file
         certFile: /path/to/client.crt
         keyFile: /path/to/client.key
```

Работа с HTTP прокси

В большинстве производственных сред прямой доступ к Интернету ограничен. Они либо не подключены к Интернету, либо работают через HTTP или HTTPS-прокси. В среде OpenShift это определение прокси-машины устанавливается как переменная среды. Это можно сделать, добавив определение прокси-сервера в файлы master и node, расположенные в /etc/sysconfig . Это похоже на то, что мы делаем для любого другого приложения.

```
Master
/etc/ sysconfig / OpenShift-master
Node
/etc/ sysconfig / OpenShift-node
   HTTP_PROXY=http://USERNAME:PASSWORD@172.10.10.1:8080/
   HTTPS_PROXY=https://USERNAME:PASSWORD@172.10.10.1:8080/
   NO_PROXY=master.vklnld908.int.example.com
```

После этого нам нужно перезапустить главный и узловой компьютеры.

```
Docker Pull
/etc/sysconfig/docker
HTTP_PROXY = http://USERNAME:PASSWORD@172.10.10.1:8080/
HTTPS_PROXY = https://USERNAME:PASSWORD@172.10.10.1:8080/
NO_PROXY = master.vklnld1446.int.example.com
```

Чтобы запустить модуль в прокси-среде, это можно сделать с помощью:

```
HTTP_PROXY=http://USERNAME:PASSWORD@172.10.10.1:8080/
HTTPS_PROXY=https://USERNAME:PASSWORD@172.10.10.1:8080/
```

OpenShift Storage с NFS

В OpenShift концепция постоянных томов и утверждений о постоянных томах образует постоянное хранилище. Это одна из ключевых концепций, при которой сначала создается постоянный том, а затем утверждается тот же том. Для этого нам необходимо иметь достаточно емкости и дискового пространства на базовом оборудовании.

Далее с помощью команды OC create создайте постоянный том:

```
$ oc create -f storage-unit1.yaml
   persistentvolume " storage-unit1 " created
```

Утверждение созданного объема:

```
   apiVersion: v1
   kind:
   PersistentVolumeClaim metadata:
      name: Storage-clame1 spec:
      accessModes:
      - ReadWriteOnce resources:
         requests:
            storage: 5Gi
```

Создайте заявку:

```
$ oc create -f Storage-claim1.yaml
persistentvolume " Storage-clame1 " created

   apiVersion: v1
   kind:
   PersistentVolume
   metadata:
      name: storage-unit1 spec:
         capacity:
            storage: 10Gi
            accessModes:
            - ReadWriteOnce nfs:
               path: /opt server: 10.12.2.2
               persistentVolumeReclaimPolicy: Recycle
```

Ключевые слова: Горизонтальное масштабирование, стратегии деплоя и выкатки приложений, администрирование узлов openshift и их сервисов в операционной системе.