Red Hat OpenShift Container Platform delivers enterprise-grade application containers

The container- and Kubernetes-based orchestration solution can lend speed and efficiency to your application performance

Your customers expect world-class service. If a customer has to wait too long for your webpage to load, or for your server to process information, you might lose a sale. To keep up with user demand, your business applications need to provide service quickly, even during peak hours.

Containers can give your applications the speed and consistency they need to satisfy your user base—and with the right container application platform, your development team can build new applications quickly to attract new customers and generate more revenue.

Red Hat® OpenShift Container Platform is a management and automation solution that enables your development teams to be productive with container technology from the start. In the Principled Technologies datacenter, we found that containers managed by OpenShift enabled an application to process over 15 thousand user requests per second. The application also had a response time as low as 4 milliseconds. Being able to fulfill requests with low latency can help you cater to your customers' needs before they decide to take their business elsewhere.

Explore this report to learn about the benefits you could gain from using OpenShift to manage and orchestrate containers in your environment.

UP TO
15,244
requests per second from a single app

Application response times as low as
4ms

How does OpenShift help you take advantage of Linux containers?

Without a platform like OpenShift, it would be difficult to use containers in a large-scale environment. OpenShift makes it feasible for companies to reap the speed and efficiency benefits of a container-based environment by automating many processes. Additionally, OpenShift provides software-defined storage and networking capabilities to support a broad range of cloud-native and stateful apps, and to connect these applications together.

An integrated blend of technologies

Red Hat OpenShift Container Platform brings together technology from several open source projects to create an enterprise-grade solution, capable of deploying and managing many applications. In addition to Linux containers and Kubernetes® for cluster management and automation, OpenShift Open vSwitch® to provide SDN for container communication inside the cluster, as well as Red Hat Cloud Forms to provide a unified management experience for infrastructure and operations teams across containers and the underlying bare-metal, virtual or cloud infrastructure. It also provides a variety of storage options for persistent application data, and an internal container registry to store and manage container images for deployment. For example, application and datacenter administrators can use these capabilities to create reproducible workflows to enhance the software development lifecycle (SDLC) and speed application delivery.

Open source Linux containers

Containers bundle software applications with everything they need to run (such as application dependencies on specific versions of system libraries), ensuring the apps will behave the same way regardless of the operating environment. The OpenShift Container Platform uses an Open Container Initiative (OCI)-compatible runtime for container execution.

With containers, you don't have to worry about setting up and maintaining different environments for each programming language or about installing the exact configurations to other environments. Your applications just run, without the headache.

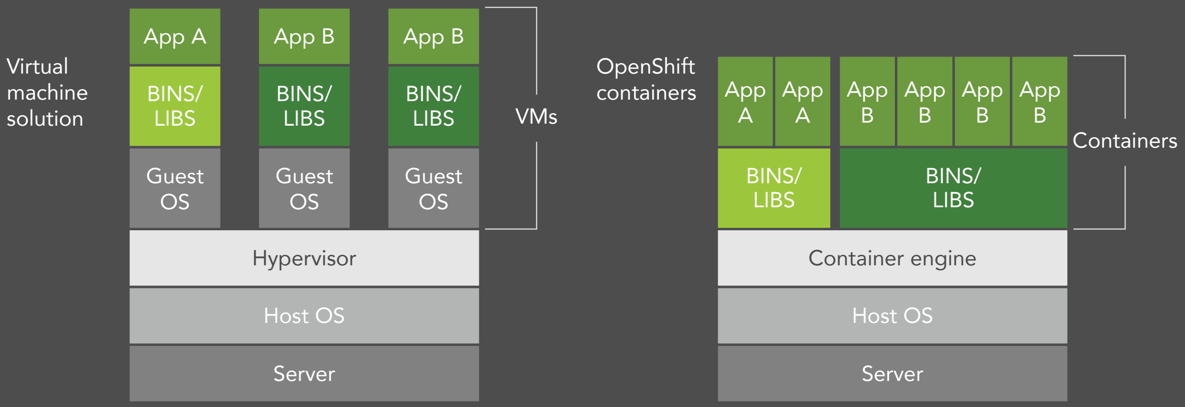
Kubernetes

OpenShift is Red Hat's supported distribution of Kubernetes, a container orchestration solution. OpenShift provides many extensions and enhancements for Kubernetes that help you build, deploy, and control Linux containers. With Kubernetes as its foundation, OpenShift provides the networking, routing, scaling, health checking, persistent storage connection, and other features that make it feasible to use containers in the enterprise.

Virtual machines (VMs) vs. containers

Functionally, containers and VMs share a goal: run isolated applications and maximize the use of physical server resources.

Virtual machines require a full system's software configuration. Each VM needs its own individual copy of the OS, binary programs (bins), libraries, and applications, which all take up their own storage, memory, CPU, and other system resources. Containers, however, share a base operating system, and have direct access to OS scheduling and resources. Each container uses fewer system resources for basic operational overhead and can apply those resources toward improving application performance.



Putting OpenShift to the test

Containers have emerged as a new way to develop, package, and deploy applications at scale. We wanted to test the performance capabilities of container-based environments under Red Hat OpenShift Container Platform.

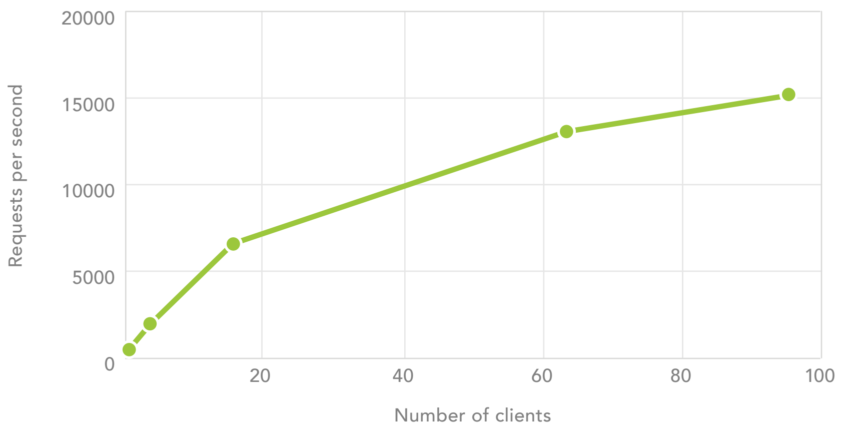
In the Principled Technologies datacenter, we set up servers running OpenShift, and tested for speed and application performance using a benchmarking application called DayTrader. As the name suggests, DayTrader simulates a high-volume day for a company's stock-trading application. Note that we deployed DayTrader with JBoss Enterprise Application Platform (EAP) as the underlying JEE application server, and PostgreSQL as the database backend.

Containers and user interactions

On any given day, your enterprise applications likely process a high volume of requests from many different users and systems. That number can spike for unexpected reasons, so you'll want a solution that can appropriately handle unpredictable workload demands.

We measured how many user interactions per second DayTrader processed in the OpenShift environment. We tested a single two-tier DayTrader application on a single OpenShift node (host). For this test, DayTrader running on OpenShift processed up to 15,244 requests per second.

UP TO
15,244
requests per second



More on DayTrader

DayTrader is a benchmark that makes a large number of user requests to simulate a high-volume day for a company's stock-trading app. These requests are things like clicking through to another page, entering information into a form, and of course, buying and selling stocks.

Here is a request breakdown of our DayTrader runs:

30% – Quote Request
20% – Home Page Request
10% – Account Page Request
10% – Portfolio Page Request
4% – Buy Request
4% – Sell Request
2% – Update Profile Request
1% – Register Request
4% – Logout Request
7% – Quote Request
3% – Account Request

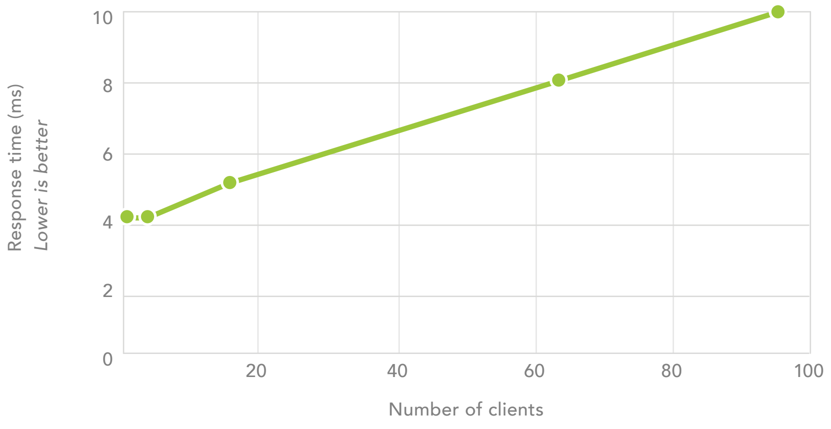
You'll notice that buying and selling stock makes up a small percentage of our test. That falls in line with typical user behavior for interactive commerce-based applications.

Think of the process of making a purchase online. Customers interact with the ordering application many times before completing the order process. They browse around pages to find the items they want, add them to their cart, and then input their information. If any one of those steps takes longer than it should, the business risks losing these customers due to a poor experience. If customers stay happy throughout their journey, however, they will be more likely to remain loyal patrons of the business.

Containers and application response times

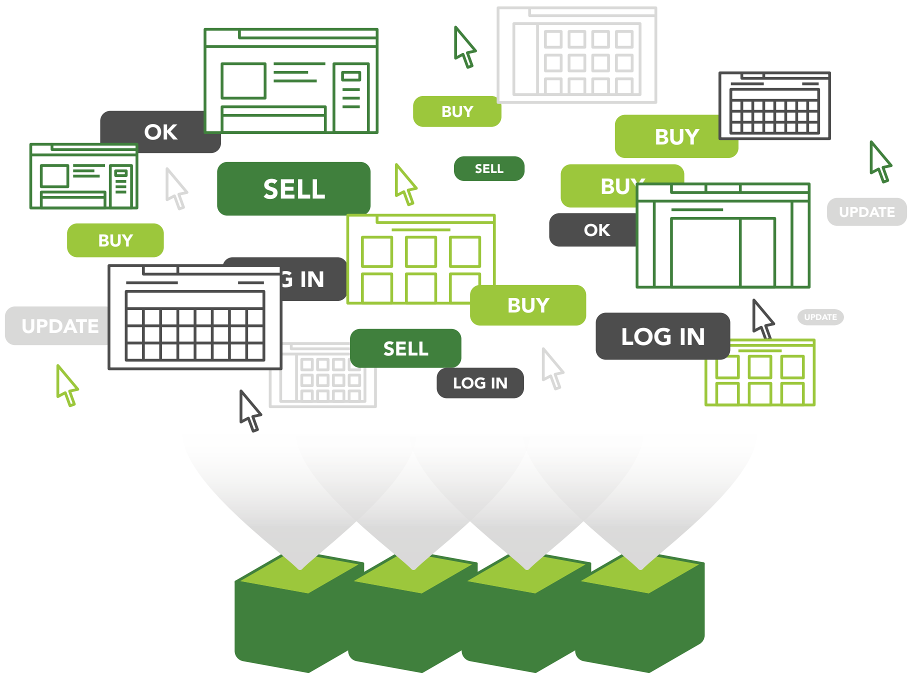
Being able to handle a heavy load of customer requests doesn't mean much if your customers are forced to wait for a sluggish server to respond. Fortunately, applications deployed on OpenShift are anything but sluggish. In our hands-on evaluations, applications in containers managed by OpenShift consistently responded as quickly as four milliseconds. Our work suggests enterprise-level applications on OpenShift are fast enough to take care of your customers—even on a busy, high-traffic day.

Application response times as low as **4ms**



Conclusion

In our hands-on testing, Red Hat OpenShift Container Platform enabled an application to process many user interactions and respond quickly to application requests. The high request rates and low response times afforded by the container-based OpenShift solution are well suited for enterprise businesses with a need for serious application performance.



On May 23, 2017, we finalized the hardware and software configurations we tested. Updates for current and recently released hardware and software appear often, so unavoidably these configurations may not represent the latest versions available when this report appears. For older systems, we chose configurations representative of typical purchases of those systems. We concluded hands-on testing on May 23, 2017.

Appendix A: System configuration information

Server configuration information	**Compute Node Lenovo Rack Server RD550**	**Manager Node Lenovo Rack Server RD540**	**Client Harness Lenovo Rack Server RD630**
BIOS name and version	PB1TS395	PB1TS395	PB2TS395
Non-default BIOS settings	Default	Default	Default
Operating system name and version/build number	Red Hat Enterprise Linux® Server 7.3	Red Hat Enterprise Linux Server 7.3	Red Hat Enterprise Linux Server 7.2
Date of last OS updates/patches applied	23 February 2017	23 February 2017	23 February 2017
Power management policy	Max. Performance	Max. Performance	Max. Performance
Processor			
Number of processors	2	2	2
Vendor and model	Intel® Xeon® E5-2699 v3	Intel Xeon E5-2690 v2	Intel Xeon E5-2690 v3
Core count (per processor)	18	10	12
Core frequency (GHz)	2.3	3.0	2.6
Stepping	C1	M1	M1
Memory module(s)			
Total memory in system (GB)	128	128	128
Number of memory modules	8	8	8
Vendor and model	SK Hynix® HMA42GR7MFR4N-TF	SK Hynix HMA42GR7MFR4N-TF	SK Hynix HMA42GR7MFR4N-TF
Size (GB)	16	16	16
Type	RDIMM	PC4-2133	RDIMM
Speed (MHz)	2,133	2,133	2,133
Speed running in the server (MHz)	2,133	2,133	2,133
Storage controller			
Vendor and model	Lenovo® 720i	LSI MEGARAID SAS 9270-8I	Lenovo 720ix
Cache size (GB)	1	1	1
Firmware version	125.03.05.01	23.12.0-0020	17.06
Driver version	lsi_mr3, 6.910.18.00-1vmw.650.0.0.4564106	lsi_mr3, 6.910.18.00-1vmw.650.0.0.4564106	megaraid_sas 06.811.02.00-rh1

Server configuration information	Compute Node Lenovo Rack Server RD550	Manager Node Lenovo Rack Server RD540	Client Harness Lenovo Rack Server RD630
Local storage			
Number of drives	3 (RAID5)	4 (RAID5)	2 (RAID1)
Drive vendor and model	Lenovo SL10A28366	Savvio ST300MM0006	Intel SSDSC2BB240G4L
Drive size (GB)	400	300	240
Drive information (speed, interface, type)	SSD, SAS	10K RPM, SAS	SSD, SATA
Network adapter: application network			
Vendor and model	Chelsio T520-CR	Mellanox MT27500	Mellanox MT27500
Number and type of ports	1x10GbE SFP+	1x10GbE SFP+	1x10GbE SFP+
Driver version	mlx4_en, 2.2-1 (Feb 2014)	cxgb4, 2.0.0-ko1	mlx4_en, 2.2-1 (Feb 2014)
Cooling fans			
Vendor and model	Delta GFM0412SS	AVC DB04056B12U	Delta PFR0612UHE
Number of cooling fans	8	8	6
Power Supplies			
Vendor and model	Delta DPS-750AB-21	Delta DPS-800RB	Delta DPS-1100EB
Number of power supplies	2	2	2
Wattage of each (W)	750	800	1,100

Appendix B: How we tested

We installed identical configurations of Red Hat OpenShift Enterprise 3.1 (now known as OpenShift Container Platform) on bare-metal servers.

Preparing the Cluster for OpenShift Enterprise

1. Reset the BIOS configuration on each server to the default.
2. Set the server power configuration to Maximum Performance.
3. Enable Hyperthreading for both processors.
4. Configure four identical SAS drives on each server into one RAID 1 volume.
5. Use one 10Gbe port on each server. Assign the first to VLAN 131 for intracluster communication, application access to the server, and management access to the server.

Installing Red Hat OpenShift Enterprise 3.4.1 on the cluster

(Note: we closely followed the OpenShift Enterprise 3.4.1 installation instructions.)

1. Install Red Hat Enterprise Linux 7.3 with its "minimal install" configuration on each server's RAID volume. Partition the volume into a boot partition, an OS LVM partition, and one 100GB partition for use by Docker.
2. On each server, register Red Hat subscription servers and attach the OpenShift Enterprise subscription.

```
subscription-manager register --username=MyUsedID --password=MyPassWord
subscription-manager attach --pool=myOSE_PoolID
```

3. On each server, configure the package manager to use these three repositories.

```
subscription-manager repos --disable="*"
subscription-manager repos --enable="rhel-7-server-rpms" \
    --enable="rhel-7-server-extras-rpms" --enable="rhel-7-server-ose-3.1-rpms"
```

4. On each server, install additional recommended packages, update the system, and reboot.

```
yum -y install wget git net-tools bind-utils iptables-services bridge-utils bash-completion ntp
yum -y update
yum install atomic-openshift-utils
systemctl reboot
```

5. On each server, create a volume group for Docker on the 100GB partition.

```
pvcreate /dev/sdb3
vgcreate docker-vg /dev/sdb3
```

6. On each node, install Docker, and initialize its local storage.

```
cat << EOF > /etc/sysconfig/docker-storage-setup

VG=docker-vg

EOF

docker-storage-setup
```

7. On each node, enable and start the chrony daemon.

```
systemctl start  chrony
systemctl enable chrony
```

8. On each node, install the appropriate SSH public keys.

```
for i in node0{1..6}; do
  ssh-copy-id –i ssh.rsa.pub $i
done
```

9. On the master node, create one set of SSH keys, and distribute the public key to each node.
10. On the OpenShift master node, install the OpenShift utilities package.

```
yum -y install docker
```

11. Update the DNS server for the master node to add a wild card entry so that any FQDN request in the domain apps.example.com will resolve to the IP address for the master node.
12. Create an ansible hosts file (ocp-hosts) with the following data:

```
[OSEv3:children]

masters

nodes
```

```
[OSEv3:vars]
ansible_ssh_user=root
debug_level=2
deployment_type=openshift-enterprise
openshift_release=v3.4
openshift_master_identity_providers=[{‘name’: ‘htpasswd_auth’, ‘login’: ‘true’, ‘challenge’: ‘true’,
‘kind’: ‘HTPasswdPasswordIdentityProvider’, ‘filename’: ‘/etc/origin/master/htpasswd’}]
openshift_master_cluster_hostname=master.example.com
openshift_master_cluster_public_hostname=master.example.com
openshift_master_default_subdomain=apps.example.com
osm_default_node_selector=’env=user’
openshift_hosted_router_selector=’env=infra’
openshift_hosted_router_replicas=1
openshift_hosted_manage_router=true
openshift_hosted_registry_selector=’env=infra’
openshift_hosted_registry_replicas=1
openshift_hosted_manage_registry=true
os_sdn_network_plugin_name=’redhat/openshift-ovs-multitenant’

[masters]
master.example.com

[nodes]
master.example.com openshift_schedulable=True openshift_node_labels=”{‘region’: ‘primary’, ‘zone’:
‘default’, ‘env’: ‘infra’}”
node01.example.com openshift_node_labels=”{‘region’: ‘primary’, ‘zone’: ‘default’, ‘env’: ‘user’}”
```

13. Install OpenShift Enterprise on the cluster by running the BYO playbook.
`ansible-playbook -I ocp-hosts /usr/share/ansible/openshift-ansible/playbooks/byo/config.yml`

Installing the DayTrader application on OpenShift

1. Create a new OpenShift project.

   ```
   oc new-project p1
   ```

2. Create a file with a certificate to provide authentication services for each pod. The contents of this file are as follows:

```
     {
 "kind": "List", "apiVersion": "v1", "metadata": {},
 "items": [
   { "kind": "ServiceAccount", "apiVersion": "v1", "metadata": { "name": "eap-service-account" },
      "secrets": [ { "name": "eap-app-secret" } ] },
   { "kind": "Secret", "apiVersion": "v1",
     "metadata": { "annotations":
       { "description": "Secret file with name 'jboss' and password 'mykeystorepass'" },
       "name": "eap-app-secret" },"data": {
       "keystore.jks":
"/u3+7QAAAAIAAAABAAAAAQAFamJvc3MAAAFNbVtLLAAABQMwggT/MA4GCisGAQQBKgIRAQEFAASCBOsxl4wqa+E+XP8+qMZY9XLhvKr
RX8V1MHdwFZQaLTEVURCizqYXoMnbhtfV0oMAUFsE7013TTA9Q2l+pSs+cqz6HH/vwjEEIkqJx5wD8WcD/bu9e9F9EHQ+zrjZFmpMFvX
svj9+ux1o/YLBDGY3kd4MoDcJy0yJ/ZpzNYLkXanlrMhWqxC7MAliCBsdyVgNn5RFb4Nn+JZgJuNSIGo/K292+0IFaFv9vsXbX889W9H
PCvfO0mQIzoy8In0NhzdKli/67y4kbDkWaI0fRONckZTxNpxn6rMc0nN9zKrGVToLxj1Ufcoj/tCvR8agtPpv7KIWUqBYDg83ad+i4EE
5XYISovlsl6RmtrrTb39PJcL86+wJ+x2ZrLuyzh6C9sAOdSBiKt/DY97ICIYltRMrb+cNwWdnJvT+PeYvv3vKo7YThha+akoJDjsWMp1
HWpbIC9zg9ZjugU+/ao6nHtmoZmCaYjLuEE+sYl5s179uyQjE3LRc+0cVY2+bYCOD6P6JLH9GdfjkR40OhjryiWy2Md6vAGaATh6kjjr
eRHfSie4KCgIZx9Ngb1+uAwauYSM8d9OIwT5lRmLd4Go9CaFXtFdq/IZv3x5ZEPVqMjxcq0KXcs1QcfK3oSYL/rrkxXxKFTrd0N3Kgvw
ATWx/KS90tdHBg65dF3PpBjK1AYQL3Q7KV3t45SVyYHd92TUsaduY1nUQk4TukNC8l9f8xYVeOFXoFHZRx9edqn8fjDMmCYn5PTPNuMP
HQm7nKxeWhV2URY5jt774gmvHLNcXeEgrM7US81wOvs2y1jY/paJWn+OACf2x2a75MWFFkZH67bZoh9pPWAwOUEtegXTL5QVicHjzZro
p8Qb7K7hlGgD0RP5YYOFYF4DD+SL5BHKr6fw/LS6MMJaK1wKsJd0oGg9HcHXjph9Kb+mqXrQ54C1KI42LpFftU3DCg8wGoqvg/zO/UtV
eHX3rBZDUIkeQrCULEkki9oL5diDxe9mNx9Qua5FJ6FJGIffQmsC4b0+Xys6NyqUu1aeWLcAPA/5hcs6ZTiSRTHTBe3vxapyBjnAL5ui
j4ILbWbEGH1e0mAHBeiihRx+w4oxH4OGCvXOhwIDHETLJJUcnJe1CouECdqdfVy/eEsIfiEheVs8OwogJLiWgzB7PoebXM4SKsAWL3NcD
tC1LV3KuPgFuTDH7MjPIR83eSxkKlJLMNGfEpUHyg+lm7aJ98PVIS+l1YV9oUzLfbo3S6S2sMjVgyviS90vNIPo5JOTEFHsg5aWJNHL0
OV4zRUeILzwwdQz+VkTk9DobnkLWUeLnwUNWheOpaQh79Mk0IfwfLj4D0Vx9p+PShKKZCGs0wjckmCFBM5Pc1x2lwMdaP5yATzrw+jUc
+/3UY4PF/4Ya66m/DRsBKEcXjVAHcTce6OdNdGlBNT8VgkxPiylwO8hvyvpf6j+wdb9iXi6eOnk0AiEJ6mUAXs/eyDD/cqQjnUBKRGLQ
USdHhvtpw8RfvyVhAAxNOnBsOT0WYol9iK6pSclGTF5mZleASRzZhH69GgdebfFhXimb0j/wYj3uLgf6mrKMDwlrXJ80SiWkXxd5TX/7
XtB9lbPzNpaR12M8U8UVg16VOtMwCR2Gss2vmhqQnQFLsUsAKcYM0TRp1pWqbzpGebCvJkVWiIYocN3ZI1csAhGX3G86ewAAAAEABVgu
NTA5AAADeTCCA3UwggJdoAMCAQICBGekovEwDQYJKoZIhvcNAQELBQAwazELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAk5DMRAwDgYDVQQH
EwdSYWxlaWdoMRYwFAYDVQQKEw1teWNvbXBhbnkuY29tMRQwEgYDVQQLEwtFbmdpbmVlcmluZzEPMA0GA1UEAxMGanNtaXRoMB4XDTE1
MDUxOTE4MDYxOFoXDTE1MDgxNzE4MDYxOFowazELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAk5DMRAwDgYDVQQHEwdSYWxlaWdoMRYwFAYD
VQQKEw1teWNvbXBhbnkuY29tMRQwEgYDVQQLEwtFbmdpbmVlcmluZzEPMA0GA1UEAxMGanNtaXRoMIIBIjANBgkqhkiG9w0BAQEFAAOC
AQ8AMIIBCgKCAQEAk0zbGtem+If//jw0OTszIcpX4ydOCC0PeqktulYkm4pG0qEVBB+HuMj7yeTBc1KCDl2xm+Q6LPeTzUufk7BXFEg4
Ru1l3PSW70LyJBfHy5ns0dYE5M1I0Avv9rvjgC1VTsiBmdXh+tIIQDPknIKpWpcs79XPOURGLvuGjfyj08EZWFvAZzYrk3lKwkceDHpY
Yb5i+zxFRz5K6of/h9gQ9CzslqNd7uxxvyy/yTtNFk2J797Vk3hKtbiATqc9+egEHcEQrzADejPYol5ke3DA1NPRBqFGku5n215i2eYz
YvVV1xmifID/3lzvNWN0bWlOxl74VsPnWa/2JPP3hZ6p5QIDAQABoyEwHzAdBgNVHQ4EFgQURLJKk/gaSrMjDyX8iYtCzPtTBqAwDQYJ
KoZIhvcNAQELBQADggEBAA4ESTKsWevv40hFv11t+lGNHT16u8Xk+WnvB4Ko5sZjVhvRWTTKOEBE5bDYfMhf0esn8gg0B4Qtm4Rb5t9P
eaG/0d6xxD0BIV6eWihJVtEGOH47Wf/UzfC88fqoIxZ6MMBPik/WeafvOK+HIHfZSwAmqlXgl4nNVDdMNHtBhNAvikL3osxrSbqdi3ey
I7rqSpb41Lm9v+PF+vZTOGRQf22Gq30/Ie85DlqugtRKimWHJYL2HeL4ywTtQKgde6JDRCOHwbDcsl6CbMjugt3yyI7Yo9EJdKb5p6Yo
VOpnCz7369W9Uim+Xrl2ELZWM5WTiQFxd6S36Ql2TUk+s8zj/GoN9ov0Y/yNNCxAibwyzo94N+Q4vA=="

     }
   }
 ]
}
```

3. Create the application:

   ```
   oc new-app eap64-postgresql-s2i -p APPLICATION_NAME=daytrader \
     -p SOURCE_REPOSITORY_URL=https://github.com/siamaksade/daytrader.git \
     -p SOURCE_REPOSITORY_REF=master -p CONTEXT_DIR=. \
     -p DB_JNDI=java:jboss/datasources/TradeDataSource -p DB_DATABASE=daytrader \
     -p DB_MAX_POOL_SIZE=1024 -p DB_MIN_POOL_SIZE=1024 -p POSTGRESQL_SHARED_BUFFERS=20GB \
     -p POSTGRESQL_MAX_CONNECTIONS=1024
   ```

Installing the Apache JMeter client on the auxiliary server

1. Download and extract Apache JMeter version 3.1.
2. Modify the file bin/jmeter to change its Java parameters. Add the following:
```
-server -d64 -XX:+UseConcMarkSweepGC -XX:+DisableExplicitGC
```
3. Modify the heap and new generation sizes to the following:
```
-Xms6g -Xmx6g -XX:NewSize=1g -XX:MaxNewSize=1g
```
4. Append the following to the file user.properties in the bin directory:
```
jmeter.save.saveservice.assertion_results_failure_message=false
jmeter.save.saveservice.assertions=false
jmeter.save.saveservice.bytes=true
jmeter.save.saveservice.data_type=false
jmeter.save.saveservice.default_delimiter=;
jmeter.save.saveservice.hostname=true
jmeter.save.saveservice.label=true
jmeter.save.saveservice.latency=true
jmeter.save.saveservice.output_format=csv
jmeter.save.saveservice.print_field_names=true
jmeter.save.saveservice.response_code=true
jmeter.save.saveservice.response_data.on_error=false
jmeter.save.saveservice.response_message=false
jmeter.save.saveservice.sample_count=true
jmeter.save.saveservice.subresults=false
jmeter.save.saveservice.successful=true
jmeter.save.saveservice.thread_counts=true
jmeter.save.saveservice.thread_name=true
jmeter.save.saveservice.time=true
jmeter.save.saveservice.timestamp_format=HH:mm:ss
summariser.interval=5
summariser.log=true
summariser.name=summary
summariser.out=true
```
5. Create the file "~/daytrader3.jmx". The contents of this file are too long to display in this section; please see the file named "daytrader3.jmx" here for reference.

Performing a DayTrader run for a single application

Below are the Java parameters we used to launch the JBoss application instance:

```
-server -XX:+UseCompressedOops -verbose:gc -Xloggc:/usr/share/jbossas/standalone/log/gc.log \
-XX:+PrintGCDetails -XX:+PrintGCDateStamps -XX:+UseGCLogFileRotation -XX:NumberOfGCLogFiles=5 \
-XX:GCLogFileSize=3M -XX:-TraceClassUnloading -Xms1303m -Xmx1303m -XX:MaxPermSize=256m \
-Djava.net.preferIPv4Stack=true -Djboss.modules.system.pkgs=org.jboss.byteman \
-Djava.awt.headless=true -Djboss.modules.policy-permissions=true \
-Dorg.jboss.boot.log.file=/usr/share/jbossas/standalone/log/server.log \
-Dlogging.configuration= file://usr/share/jbossas/standalone/configuration/logging.properties \
-jar /usr/share/jbossas/jboss-modules.jar -mp /usr/share/jbossas/modules \
-jaxpmodule javax.xml.jaxp-provider org.jboss.as.standalone \
-Djboss.home.dir=/usr/share/jbossas -Djboss.server.base.dir=/usr/share/jbossas/standalone
```

1. Run the following script with arguments equal to the number of apps to populate each app with initial data:
```
echo Setting DT configs; date

curl --data
 'action=updateConfig&RunTimeMode=1&JPALayer=1&OrderProcessingMode=0&WorkloadMix=1&WebInterface=0&MaxUs
 ers=15000&MaxQuotes=10000&markvetSummaryInterval=20&primIterations=1&EnableLongRun=on' \
   http://daytrader-p1.apps.example.com/daytrader/config > /dev/null


echo Initial PG tables; date
```

```
curl http://daytrader-p1.apps.example.com/daytrader/config?action=buildDBTables > /dev/null
curl --data \
 'action=updateConfig&RunTimeMode=1&JPALayer=1&OrderProcessingMode=0&WorkloadMix=1&WebInterface=0&MaxUs
 ers=15000&MaxQuotes=10000&marketSummaryInterval=20&primIterations=1&EnableLongRun=on' \
  http://daytrader-p1.apps.example.com/daytrader/config > /dev/null

echo Initial PG data; date

curl http://daytrader-p1.apps.example.com/daytrader/config?action=buildDB > tmp-DB-p1.txt

echo Initial PG counters; date
curl http://daytrader-p1.apps.example.com/daytrader/config?action=resetTrade

date; echo "Done"
```

2. Run the following script with arguments equal to the number of apps to perform one test of the cluster with 32 clients per app:

```
rm -f tmp-p1
sh jmeter -n -t ~/daytrader3.jmx -l OUT-p1.jtl -JHOST=daytrader-p1.apps.example.com \
  -JPORT=80 -JDURATION=300 -JTHREADS=${thread} > tmp-p1
```

This project was commissioned by Red Hat.

Principled Technologies®

Facts matter.®