## GIT+Jenkins+Docker+Ansible+Kubernetes Exercise

## Main goal

The exercise is aimed to attain the intrinsic knowledge of the basic DevOps toolkits. The stages are created in the regards to the courses curriculum so suppose to be added and run step by step during the hole program.

At the end all the students would evolve the Jenkins pipeline for getting the source code from the GIT test public repository, building the artifacts using Maven, fetching them to the tomcat-based Docker container, putting (pushing) the container image to the Docker Hub and installed Sonatype Nexus repository, getting (pulling) that image from the registry and running the container with the port exposing on the local Docker. The Kubernetes cluster has been opted as the final deployment destination for the test product.

## Initial state

Every student is supplied with the credentials for the virtual course PC and could reach it via HTTP + SSH and has an accounts:

* DockerHub (<https://hub.docker.com/>) (after the Docker course)
* GitHub (<https://github.com/>) (after the GIT basic course)

For the Kubrnetes step we need the K8S cluster to be deployed and the kubectl from the Jenkins environment to be configured as well. (after the Kubernetes course)

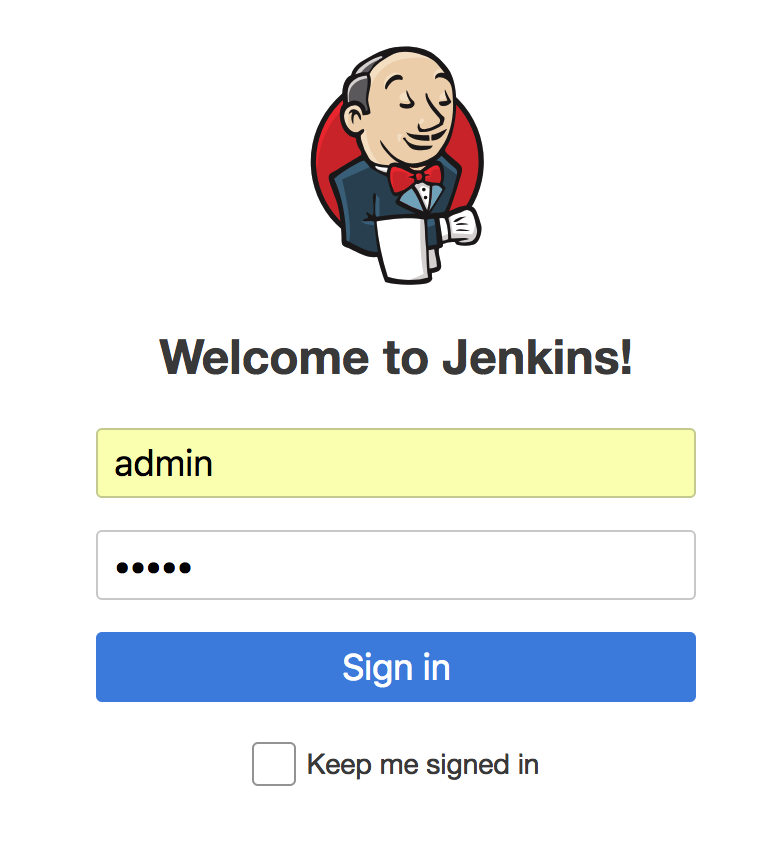
## Preliminary items

1. Get to the course Jenkins environment using the following credentials:

*http://<your course PC IP>:8080/*

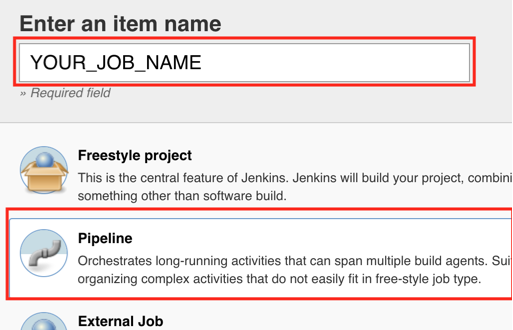
User: *admin*

Password: *admin*

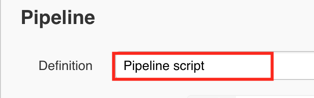


1. Create a Jenkins Pipeline job:





1. Set the pipeline definition as “Pipeline Script” and start creating the code

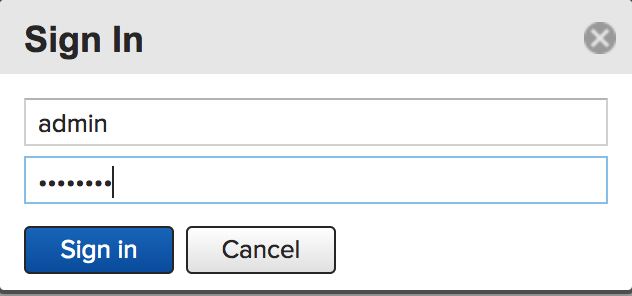


1. Create the docker repository in the Nexus

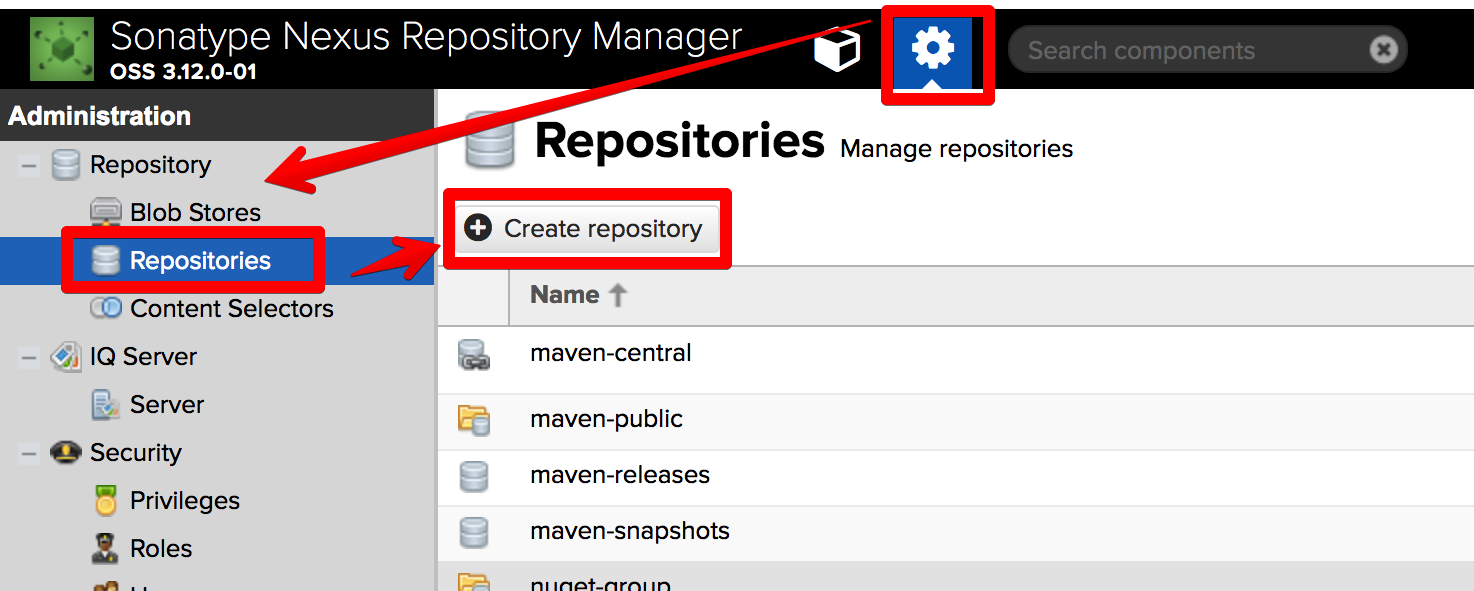
* Go to the *http://<your course PC IP>:8081/*
* Sign in under the default user and password:

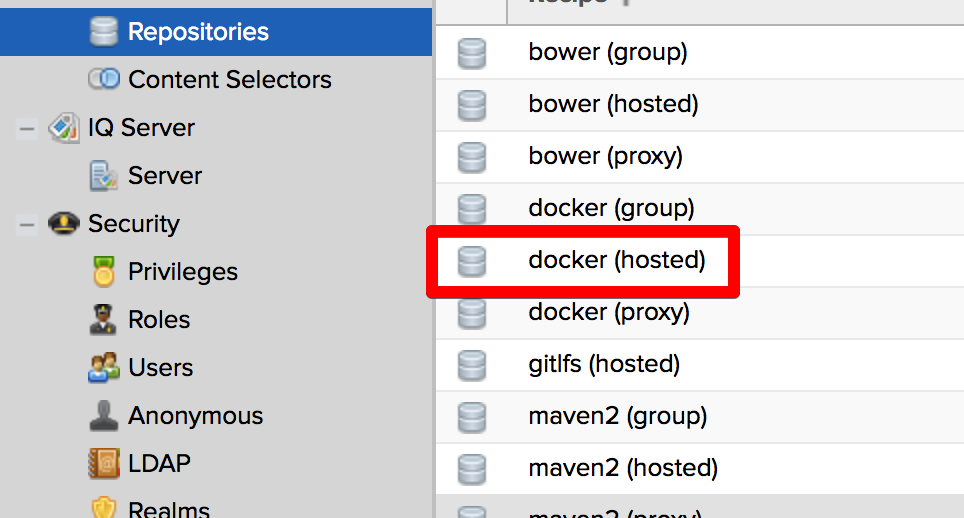


**admin/admin123**

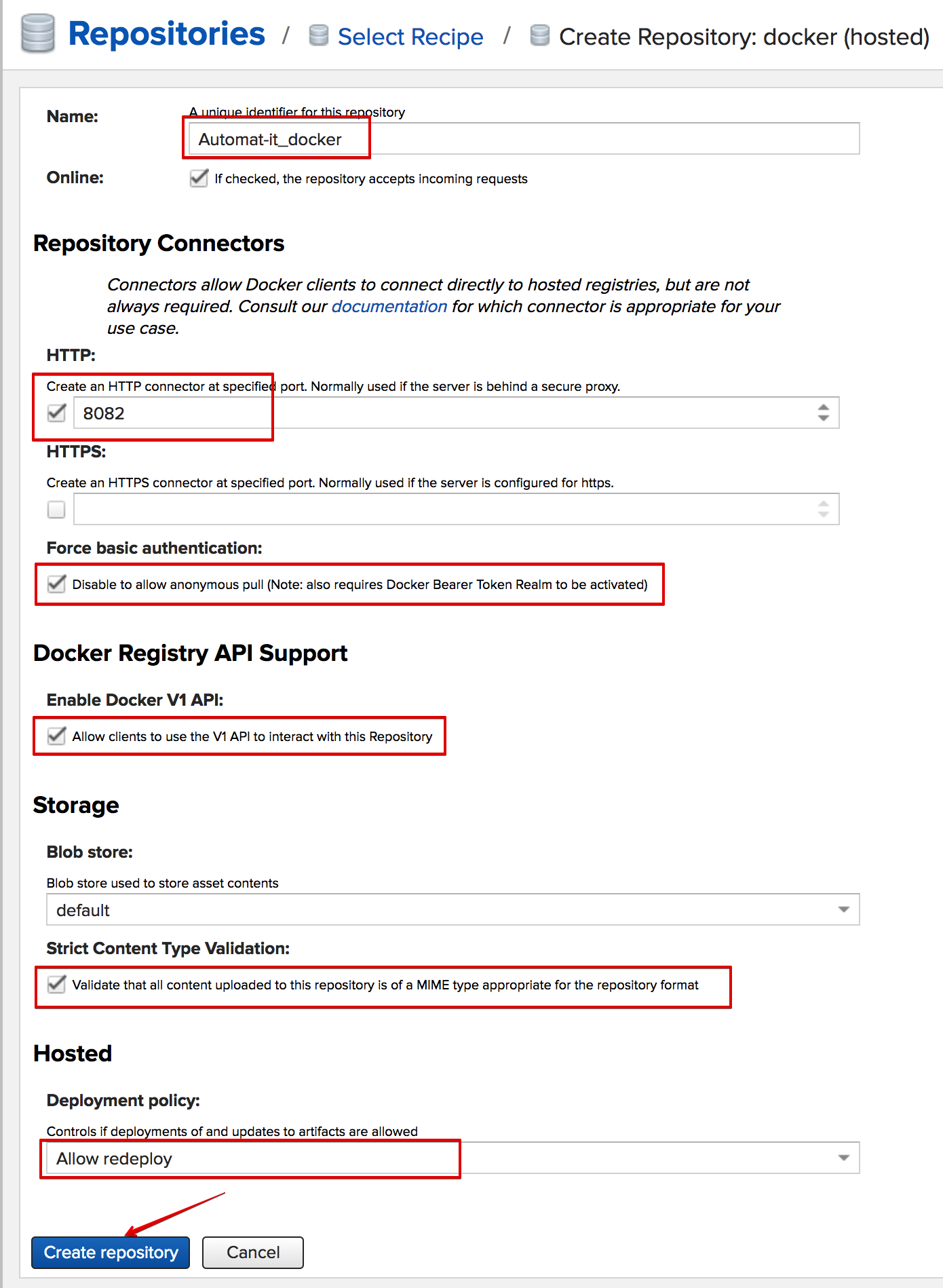


* Create the repository





Consider setting any name you do want, but please leave the port 8082, since it is configured for being reachable from the outside:



1. Set Docker to allow http repository

* Create the docker daemon override config file:

***sudo vi /etc/docker/daemon.json***

and add there the configured docker repository (please copy-paste):

***{  
 "insecure-registries" : ["127.0.0.1:8082"]  
}***

* Restart the Docker service using:

***sudo service docker restart***

1. Pair the docker with the private repository

***docker login -u 'admin' -p 'admin123' 127.0.0.1:8082***

*WARNING! Using --password via the CLI is insecure. Use --password-stdin.  
WARNING! Your password will be stored unencrypted in /home/ubuntu/.docker/config.json.  
Configure a credential helper to remove this warning. See  
https://docs.docker.com/engine/reference/commandline/login/#credentials-store* ***Login Succeeded***

## Create the Groovy pipeline with the following stages:

1. Stage ‘Preparation’:

* Define Maven home variable and connect it to the M3 alias from Global Tool Configuration
* Source the course test code (time-tracker) from the own public GitHub repository made during the “Basic of GIT” course OR from the public GIT repository: **https://github.com/zivkashtan/course.git**

1. Stage ‘Creating Package’:

* Using the predefined in stage 1 home variable, create the package with maven
* Check the result \*.war files. The files could be found via the SSH CLI on the filesystem:

***sudo su - jenkins***

***ls -ltr ~/workspace/<YOUR\_JOB\_NAME>/web/target/***

***<supervise the file>***

***exit***

1. Stage ‘Creating Dockerfile’:

On that stage we need to take the existed test tomcat-based docker image within the JRE8 installed and inject our \*.war artifact to the default tomcat folder for the artifacts (tomcat:8.0.20-jre8 has that directory there: /usr/local/tomcat/webapps/). Please do bear in mind that the tomcat-based image is a part of the official docker repository and it is no need to create it from the scratch. The repository could be supervised by drilling down the following link and typing ‘tomcat’ in the search form:

<https://hub.docker.com/explore/?page=1>

The stage items are:

- Create the Dokerfile for creating the product image based on ‘tomcat:8.0.20-jre8’

- Add to the Dokerfile the \*.war file with the product: destination folder in the tomcat is

1. Stage ‘Docker build image’:

* Run the docker build command with tag (***<your\_dockerhub\_username>/time-tracker***) to create an image in the local docker

1. Stage ‘Ansible push image’:

* Create/modify the example /home/ubuntu/hosts inventory file via SSH CLI to let the ansible work with the localhost. The file looks like the following example:

***[local]***

***localhost ansible\_connection=local***

* Create/modify the example /home/ubuntu/docker\_push\_playbook.ym file via SSH CLI to trigger the ansible do a docker login to your dockehub repository and push there the built image:

***- name: Ansible Docker PUSH step***

***hosts: localhost***

***tasks:***

***- name: Log into Docker Hub and force re-authorization***

***docker\_login:***

***username: <YOUR\_DOCKERHUB\_USERNAME>***

***password: <YOUR\_DOCKERHUB\_PASSWORD>***

***email: <YOUR\_DOCKERHUB\_EMAIL>***

***reauthorize: yes***

***- name: push an image***

***docker\_image:***

***name: <YOUR\_DOCKERHUB\_USERNAME>/time-tracker***

***tag: latest***

***push: yes***

***- name: push an image to the local Nexus registry  
 docker\_image:  
 name: <YOUR\_DOCKERHUB\_USERNAME>/time-tracker  
 repository: 127.0.0.1:8082/time-tracker  
 tag: latest  
 push: yes***

* Run command via Jenkins to apply the playbook:

***ansible-playbook /home/ubuntu/docker\_push\_playbook.yml -i /home/ubuntu/hosts***

1. Stage ‘Ansible pull and run image’:

* Create/modify the example /home/ubuntu/docker\_pull\_run\_playbook.ym file via SSH CLI to trigger ansible to pull your docker hub repository image and run it with the port exposing:

***- name: Ansible Docker step***

***hosts: localhost***

***tasks:***

***- name: pull an image***

***docker\_image:***

***name: <YOUR\_DOCKERHUB\_USERNAME>/time-tracker:latest***

***- name: Start a container***

***docker\_container:***

***name: time-tracker***

***image: <YOUR\_DOCKERHUB\_USERNAME>/time-tracker:latest***

***state: started***

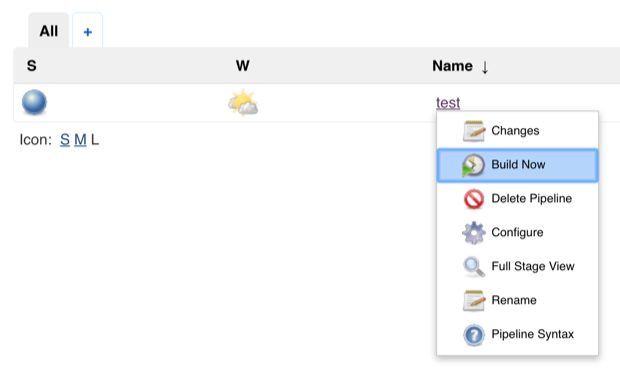
***ports:***

***- "80:8080"***

* Run command via Jenkins to apply the playbook:

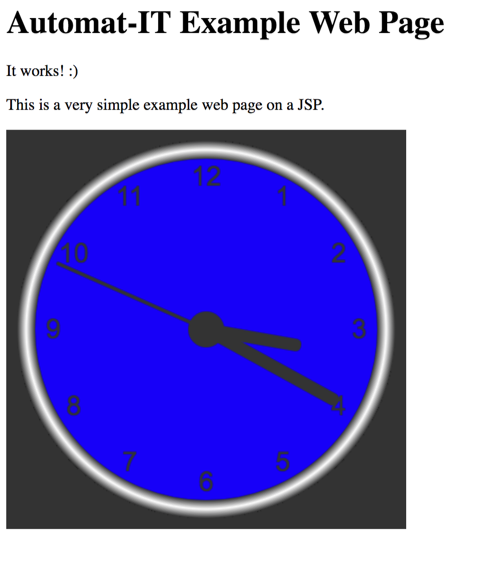
***ansible-playbook /home/ubuntu/docker\_pull\_run\_playbook.yml -i /home/ubuntu/hosts***

1. Run the pipeline and see the stdout:

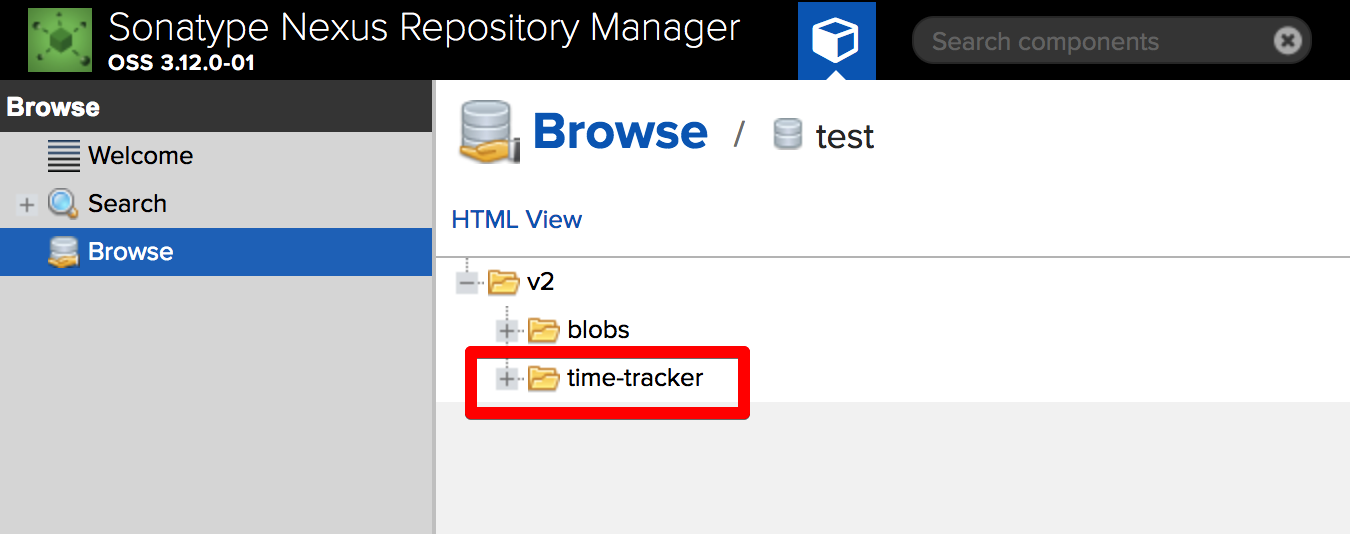


1. Verify the results using the a WEB browser:

* Go to *http://<your course PC IP>:80/time-tracker-web-0.3.1/*
* You should see the tremendous Automat-IT example web page:



1. Check the image pushed to the local Nexus private repository:



1. Switch under the jenkis user and modify the source code:

***sudo su - jenkins***

***vi ./workspace/<YOUR\_JOB\_NAME>/web/src/main/webapp/index.jsp***

Put ‘green’ instead of ‘blue’:

***...***

***function drawFace(ctx, radius) {***

***var grad;***

***ctx.beginPath();***

***ctx.arc(0, 0, radius, 0, 2\*Math.PI);***

***ctx.fillStyle = 'green';***

***ctx.fill();***

***grad = ctx.createRadialGradient(0,0,radius\*0.95, 0,0,radius\*1.05);***

***...***

1. Go to Jenkins pipeline and commit out in the Stage ‘Preparation’ the line with the git clone:

***stage('Preparation') {***

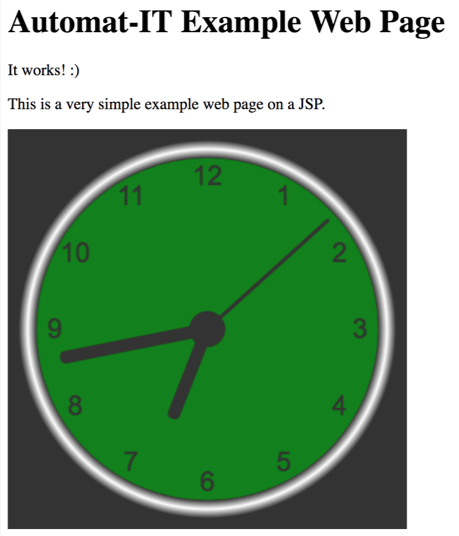
***// Get some code from a GitHub repository***

***//git 'https://github.com/zivkashtan/course.git';***

***// Get the Maven tool.***

1. Run the Pipeline and verify the results using the a WEB browser:

* Go to *http://<your course PC IP>:80/time-tracker-web-0.3.1/*
* You should see the fascinating Automat-IT example web page with green color:



1. Stage ’Kubernetes deployment’:

* Using vi (vim) create/modify the deployment yaml file in the user home directory. As the example below, the deployment name is **time-tracker-deployment.yaml** and application name is time-tracker, please copy-paste the structure and enrich within your’s credentials :

**cd ~**

**vi time-tracker-deployment.yaml**

***apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: time-tracker  
 labels:  
 app: time-tracker  
spec:  
 replicas: 3  
 selector:  
 matchLabels:  
 app: time-tracker  
 template:  
 metadata:  
 labels:  
 app: time-tracker  
 spec:  
 containers:  
 - name: time-tracker  
 image: <YOUR\_DOCKER\_HUB\_NAME>/time-tracker:latest***  
 ***ports:  
 - containerPort: 8080***

In this example:

* A Deployment named time-tracker is created, indicated by the metadata: name field.
* The Deployment creates three replicated Pods, indicated by the replicas field.
* The selector field defines how the Deployment finds which Pods to manage.
* In this case, we simply select on one label defined in the Pod template (app: time-tracker).
* The Pod template’s specification, or template: spec field, indicates that the Pods run one container, time-tracker, which runs the time-tracker Docker Hub image at latest version.
* The Deployment opens tomcat’s port 8080 for use by the Pods.
* Using vi (vim) create the service yaml file in the home directory. As the example below, the service filename is **time-tracker-LB-service.yaml** and service name is **time-tracker**:

**apiVersion: v1  
kind: Service  
metadata:  
 name: time-tracker  
 labels:  
 app: time-tracker  
spec:  
 type: LoadBalancer  
 ports:  
 - port: 80  
 targetPort: 8080  
 protocol: TCP  
 selector:  
 app: time-tracker**

In this example:

* This specification will create a new Service object named “**time-tracker**” which targets TCP port 8080 on any Pod with the "app=**time-tracker**" label to the external targetPort (By default the targetPort will be set to the same value as the port field).
* This Service will also be assigned an AWS load balancer.(type: **LoadBalancer**)
* The Service’s selector will be evaluated continuously and the results will be POSTed to an Endpoints object also named “time-tracker”.
* Kubernetes Services support TCP and UDP for protocols. The default is TCP.
* In the pipeline run the deployment and service creation:

***kubectl create -f ~/\*.yaml***

1. Check Kubernetes deployment:

Check the running service via the SSH CLI :

***kubectl get svc -o wide***

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *NAME* | *TYPE* | *CLUSTER-IP* | *EXTERNAL-IP* | *PORT(S)* | *AGE* | *SELECTOR* |
| *kubernetes* | *ClusterIP* | *10.233.0.1* | *<none>* | *443/TCP* | *38d* | *<none>* |
| *time-tracker* | *LoadBalancer* | *10.233.28.131* | *a444db75768a211e88ca2067b1f27614-1420067569.eu-central-1.elb.amazonaws.com* | *80:30008/TCP* | *14s* | *app=time-tracker* |

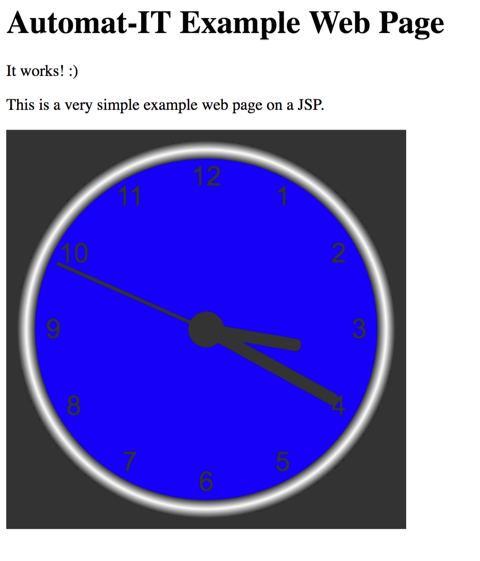
***kubectl describe svc time-tracker***

*Name: time-tracker  
Namespace: default  
Labels: app=time-tracker  
Annotations: <none>  
Selector: app=time-tracker  
Type: LoadBalancer  
IP: 10.233.28.131  
LoadBalancer Ingress: a444db75768a211e88ca2067b1f27614-1420067569.eu-central-1.elb.amazonaws.com  
Port: <unset> 80/TCP  
TargetPort: 8080/TCP  
NodePort: <unset> 30008/TCP  
Endpoints: 10.233.126.13:8080,10.233.98.13:8080,10.233.98.14:8080  
Session Affinity: None  
External Traffic Policy: Cluster  
Events:  
 Type Reason Age From Message  
 ---- ------ ---- ---- -------  
 Normal EnsuringLoadBalancer 3m service-controller Ensuring load balancer  
 Normal EnsuredLoadBalancer 3m service-controller Ensured load balancer*

**Please note**, that to get the load balancer configured we need to wait about **3-6 minutes**

* Verify the results using the a WEB browser:

Go to *http://<your LoadBalancer Ingress>:80/time-tracker-web-0.3.1/*



1. Test Kubernetes deployment:

To test the cluster we would delete all the pods from the deployment and ensure they are up afterwards

* Get the running pods by executing the below command from any node:

**kubectl get pods**NAME READY STATUS RESTARTS AGE  
time-tracker-89bdf6dc4-8vlgl 1/1 Running 0 4m  
time-tracker-89bdf6dc4-bzr2w 1/1 Running 0 3m  
time-tracker-89bdf6dc4-mxpf2 1/1 Running 0 3m

* Delete all the running pods, please note that pod names obtained on the previous step have to be separated by spaces:

**kubectl delete pod time-tracker-89bdf6dc4-8vlgl time-tracker-89bdf6dc4-bzr2w time-tracker-89bdf6dc4-mxpf2**pod "time-tracker-89bdf6dc4-8vlgl" deleted  
pod "time-tracker-89bdf6dc4-bzr2w" deleted  
pod "time-tracker-89bdf6dc4-mxpf2" deleted

* Check the new pods have been created automaticlly with different names:

**kubectl get pods**  
NAME READY STATUS RESTARTS AGE  
time-tracker-89bdf6dc4-gl42k 1/1 Running 0 1m  
time-tracker-89bdf6dc4-n68c2 1/1 Running 0 1m  
time-tracker-89bdf6dc4-qqcjt 1/1 Running 0 1m

* Verify the results using the a WEB browser:

Go to *http://<your LoadBalancer Ingress>:80/time-tracker-web-0.3.1/*