ThoughtWorks®

O'REILLY®
Velocity

Creating pipelines that build, test and deploy containerized artifacts

Slides: https://goo.gl/2mzFe6

Tom Adams

tadams@thoughtworks.com



Who I am

Tom Adams

ThoughtWorks®
Tech Lead

tadams@thoughtworks.com

http://tadams289.blogspot.com



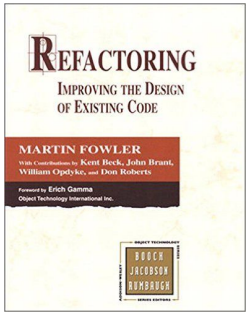
Today's Journey - Docker and Build Pipelines

- Introductions
- Lab 1 - Overview of application
- What is continuous integration?
- CI - Practices, How, Team Responsibilities
- Lab 2 - Moving application build to Build Server
- Testing Pyramid
- Docker: file, image, container
- Lab 3 - Creating Docker build artifacts (*break*)

- Why use containers as build artifacts?
- Lab 4 - Moving Docker builds to Build Server
- Lab 5 - Deploying Docker images
- Wrap-up / Take aways

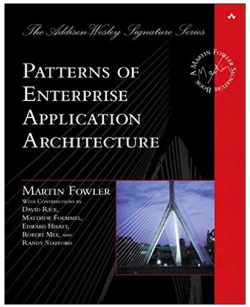
ThoughtWorks®

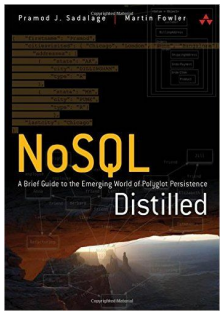




1999

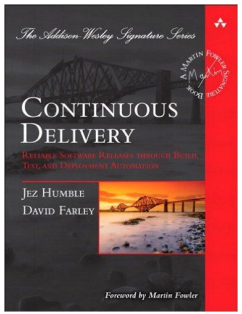
2003

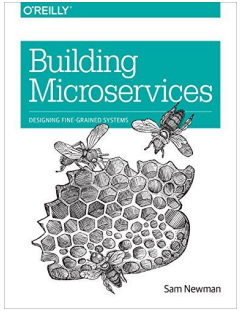




2012

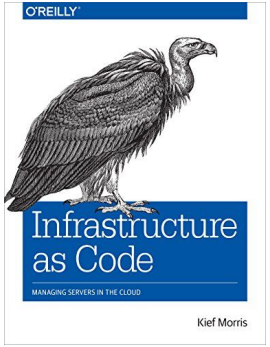
2010





2015

2016



Goals of the workshop

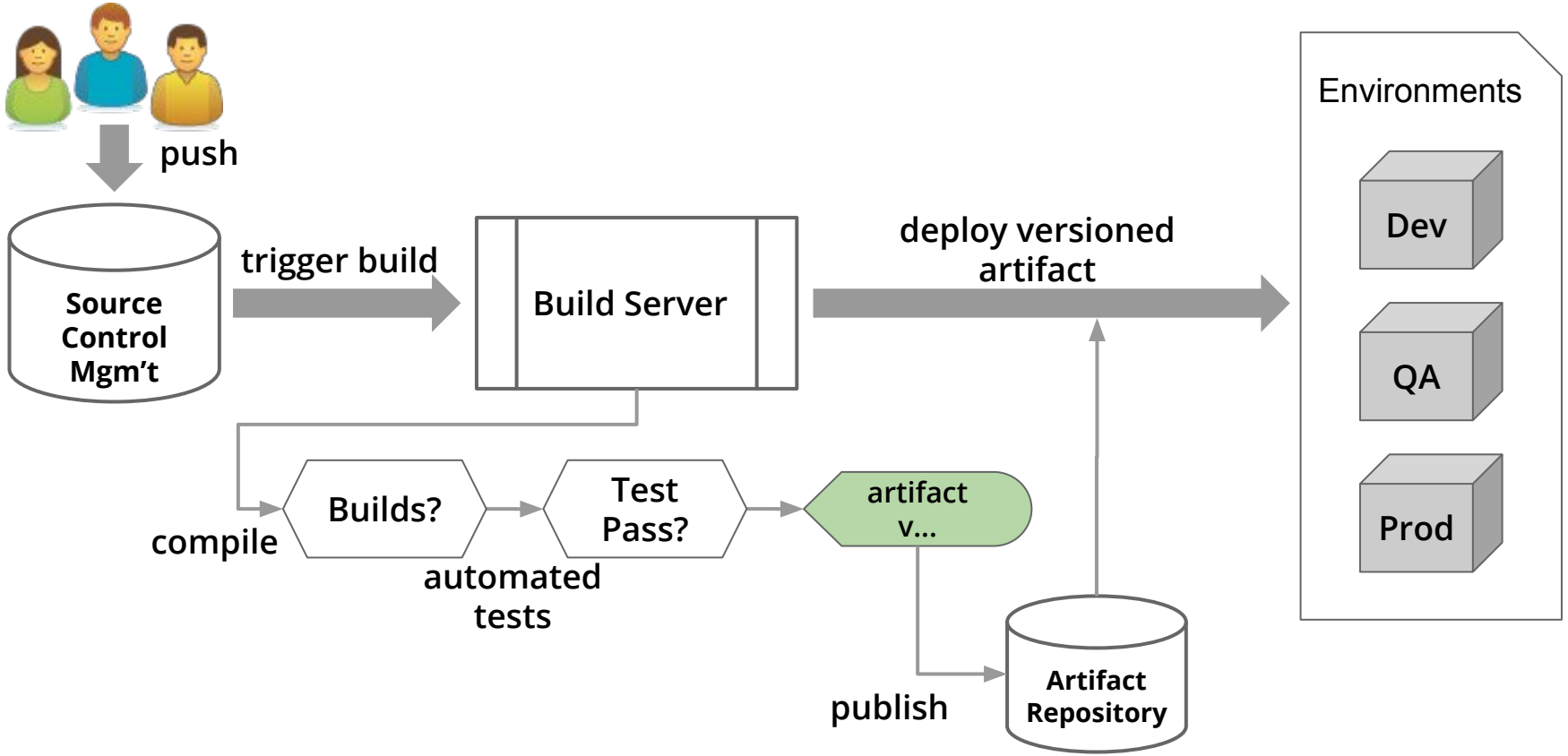
Create and deploy docker images using a build pipeline.

Docker and ...

- Continuous Integration
- Versioning
- Testing Pyramid
- Automated Deployment

Gain experience working with a build pipeline and docker.
Go from hearing / reading about it to doing it...

Build Pipeline



The App: mlb-scores



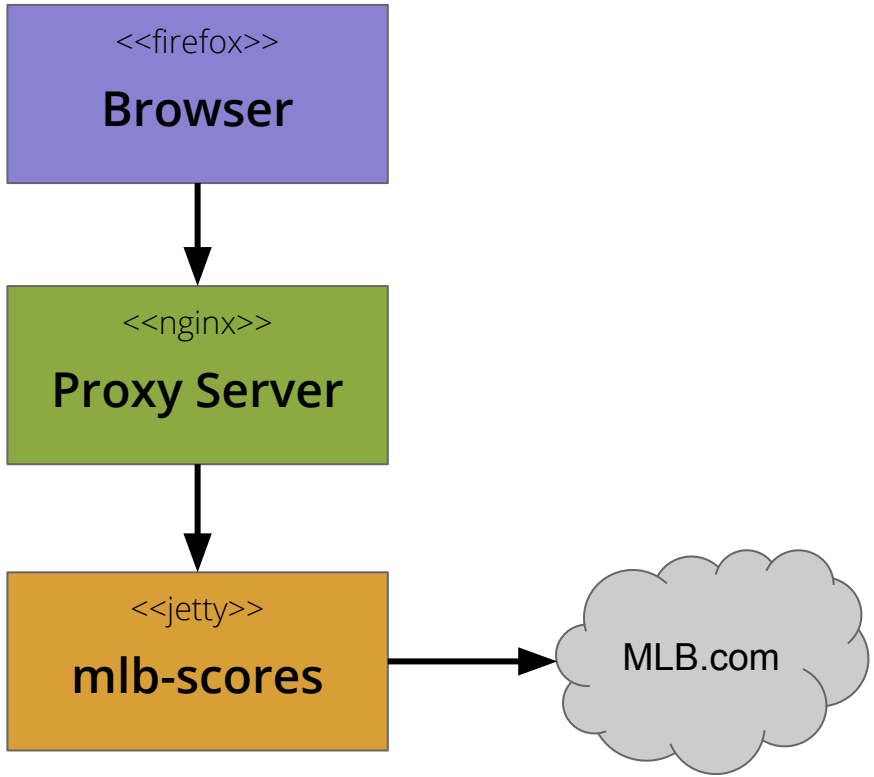
Game Date: yyyy-MM-dd **Submit Query**

Games Played: April 23, 2017

Away	Team	Runs	Home	Team	Runs
	Cubs	5		Reds	7
	Astros	6		Rays	4
	Braves	2		Phillies	5

The App: mlb-scores

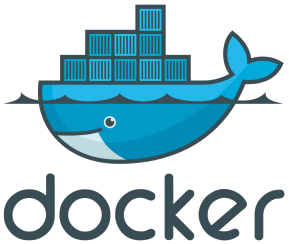
- Displays baseball scores for a given day
- NGINX used as a simple reverse proxy
- Web application is written in Java 8 with embedded Jetty server

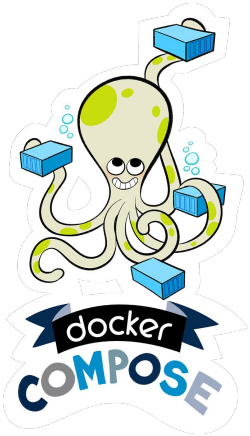


Tools / Technologies

Goals of a build pipeline are the same regardless of the tools used.





















Lab 1 - Overview of Application

Goals:

- Build / run the Java application
- Ensure VM works and has a network connection

Steps:

1. Start VM, user name is DevOps, password is “none”

2. Change directory to git repository at ~/mlb-scores

```
devops@DocCi-WrkShop:~$ cd mlb-scores/
```

3. Run gradle script to build, test and package (jar) application.

```
devops@DocCi-WrkShop:~/mlb-scores$ gradle clean build test integrationTest jar
```

Lab 1 - Overview of Application

Steps:

4. Start application from jar

```
$java -jar build/libs/mlb-scores-???.jar
. . . .
2017-09-13 20:20:53 INFO  MLBScoresApplication:51 - Server started at port 8080
```

5. Once you see the server started log message open firefox and enter the following URL:
   localhost:8080/mlb-scores/scores
6. Entering a date in the correct format will display baseball scores from that date (if available from the MLB API).

Lab 1 - Overview of Application

Steps:

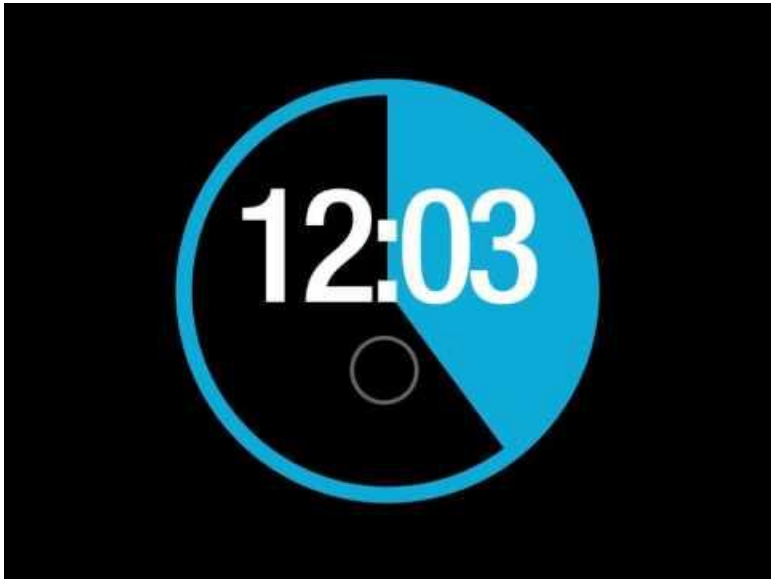
7. Open a new tab in the terminal window (ctrl-shift t)
8. Run the selenium functional tests against the running application.

```
devops@DocCi-WrkShop:~/mlb-scores$ gradle localFunctionalTest
```

9. Firefox should reflect the commands being executed by the selenium webdriver.
10. Stop the mlb-scores application using cntl-c.

Lab 1 - 2:05pm





What is continuous integration?

- Team development practice that requires code changes to be push to a shared repository several times a day.
- Each commit is built and tested by an automated build - allowing teams to detect problems early.

> *"Continuous Integration doesn't get rid of bugs, but it does make them dramatically easier to find and remove."*
>
> - Martin Fowler, Chief Scientist, ThoughtWorks

What is continuous integration?



Paul Hammant @paul_hammant · Jul 28

90% survey participants who said they were practising CI, missed the need to integrate to 'trunk' daily (paraphrase)

CI - Practices

- Maintain a single development trunk (no feature branches)
- Automate the build
- Make your build self-testing
- Every commit should build on a build server
- Keep the build fast
- Test in a simulation of the production environment
- Make it easy for anyone to get the latest executable
- Everyone can see what’s happening
- Automate deployment

CI - How

- Developers check out code into their private workspaces.
- When done, commit the changes to ***the development trunk***.
- The CI server monitors the repository and checks out changes when they occur.
- The CI server builds the system and runs unit and integration tests.
- The CI server releases deployable artifacts for testing.
- The CI server assigns a build label to the version of the code it just built.
- The CI server informs the team of the successful build.
- If the build or tests fail, the CI server alerts the team.
- The team fixes the issue ***immediately*** or reverts the commit.

CI - Team Responsibilities

- Check in frequently
- Don’t check in broken code
- Don’t check in code that does not have tests
- Don’t check in when the build is broken
- Don’t go home after checking in until the system builds (all test pass, or code is reverted)

Lab 2 - Moving build to CI Server

Goals:

- Test Jenkins build Server
- Verify application artifact is versioned

Steps:

1. In Firefox navigate to Jenkins home page - **http://localhost:9090**

2. Review the configuration for the mlb-scores_Build
   a. Click on mlb-scores_Build job
   b. Click on Configure on left hand side of page

3. The build is kicked off by a git post-commit hook defined in the `~/mlb-scores/.git/hooks/post-commit` script

Lab 2 - Moving build to CI Server

Steps:

4. Make a small change in the git repository and push it to trigger a build.
    a. Change the file at ~/mlb-scores/ReadMe.txt using vim or gedit.
    b. Commit the change using git

```
$ git add .

$ git commit -m”Test Jenkins”
Scheduled polling of mlb-scores_Build
No Git consumers using SCM API plugin for: file:///home/devops/mlb-scores
```

    c. Note that Jenkins will pick-up the commit despite the warning message

5. Check the Jenkins home page to see the build job running.
6. Navigate to the mlb-scores_Build page in Jenkins.
7. Download the jar file under latest sucessful artifacts.

Lab 2 - Moving build to CI Server

Steps

8. Start the mlb-scores application using the downloaded jar file.

```
$ cd ~/Downloads

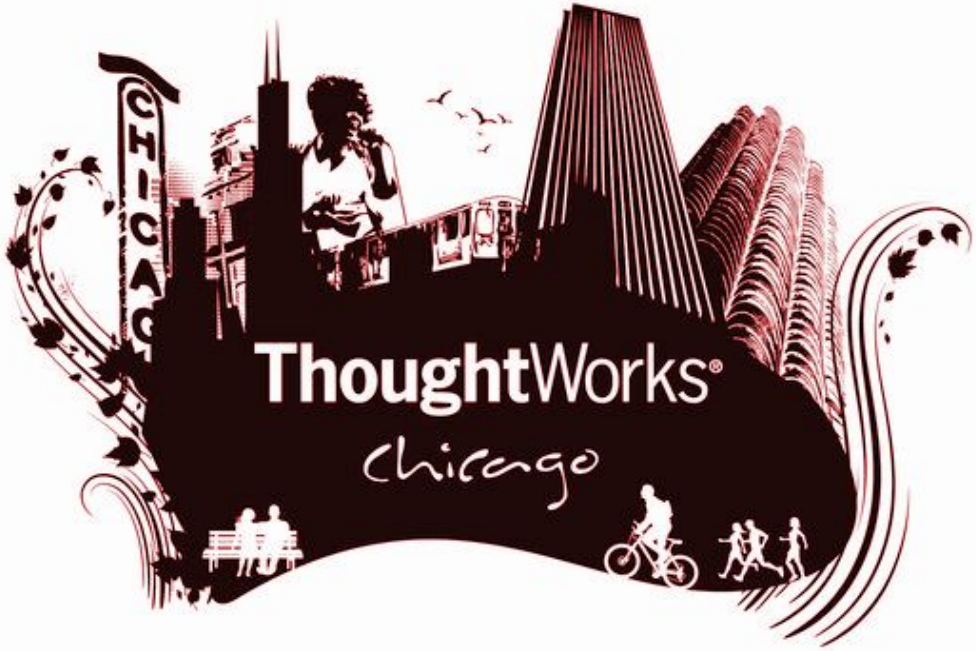
$ java -jar mlb-scores-1.???.jar
```

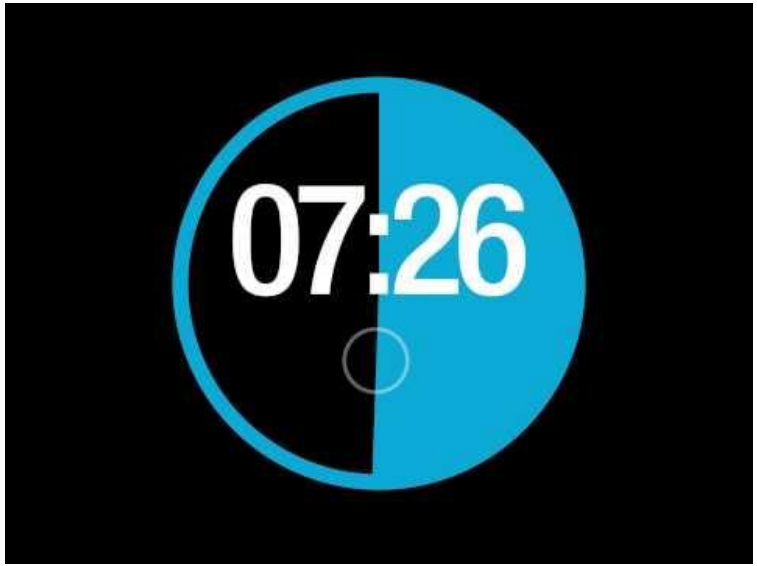
9. Check the application version at
   localhost:8080/mlb-scores/HealthCheck
10. Verify the current git describe matches the application version.

```
$ cd ~/mlb-scores

$ git describe
1-9-g165f618
```

Lab 2 - 2:35pm





Lab 3 - Building & Testing Docker images

- Creating Docker images
  - Dockerfile -> Image -> Container
  - Docker Layers
  - Docker tagging
- Automated ***testing*** of Docker images
- Running multi-container docker applications with docker-compose

Docker images and containers

- Dockerfile
  - Source code that defines a docker image
  - Managed in a version control system
- Docker Image
  - Template created by building the Dockerfile
  - Immutable - cannot be modified
  - Managed in a docker registry
- Docker Container
  - Instance of an image
  - Can be mutated
  - Managed by the docker process on a host machine



Docker Layers

- Docker images are built on a series of read only file system layers

nginx docker imag

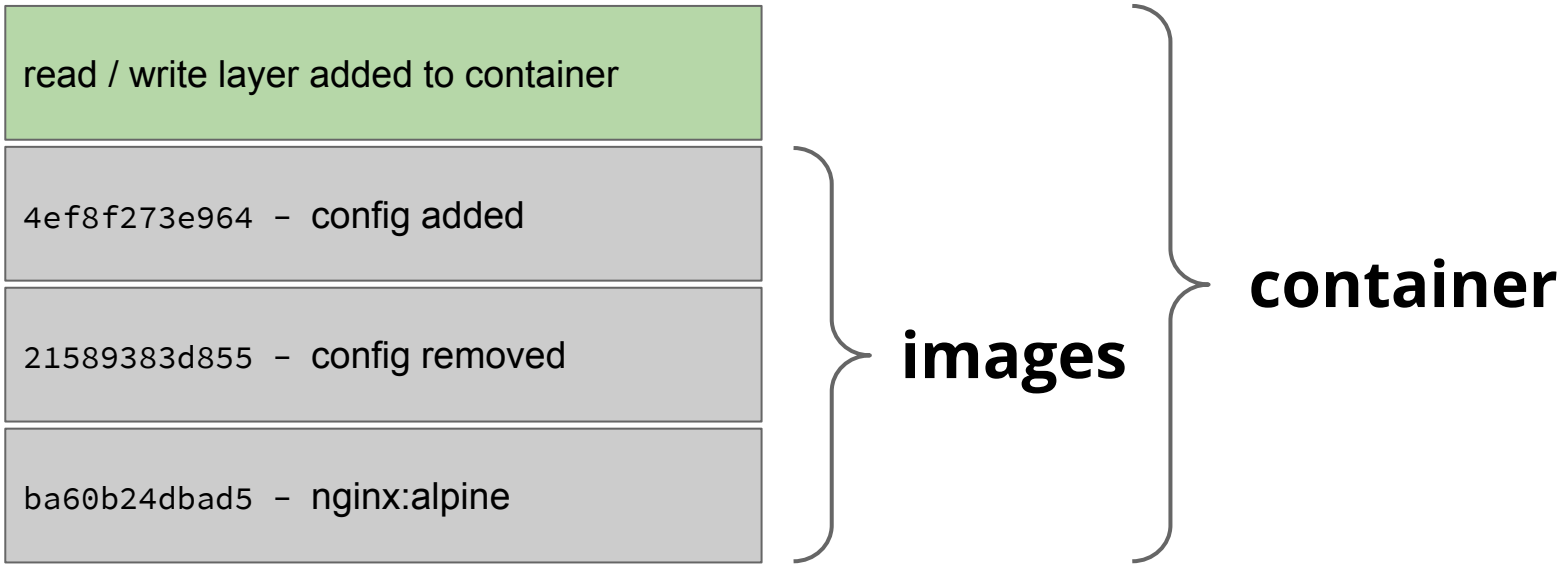
```
FROM nginx:alpine

RUN rm -v /etc/nginx/nginx.conf
ADD nginx.conf /etc/nginx/
```

```
devops@DocCi-WrkShop:~$ do                                    ginx:10
IMAGE               CREATE                                                  SIZE
4ef8f273e964        29 hours ago        /bin/sh -c #(nop) ADD file:a4b1aebf16779f7...   937B
21589383d855        30 hours ago        /bin/sh -c rm -v /etc/nginx/nginx.conf          0B
ba60b24dbad5        6 weeks ago         /bin/sh -c #(nop)  CMD ["nginx" "-g" "daem...   0B
<missing>           6 weeks ago         /bin/sh -c #(nop)  STOPSIGNAL [SIGTERM]         0B
<missing>           6 weeks ago         /bin/sh -c #(nop)  EXPOSE 80/tcp                0B
<missing>           6 weeks ago         /bin/sh -c #(nop) COPY file:1d1ac3b9a14c94...   1.09kB
<missing>           6 weeks ago         /bin/sh -c #(nop) COPY file:f94db45bb7e4b...    643B
<missing>           6 weeks ago         /bin/sh -c #(nop) ADD file:8F6F510D421...       11.5MB
<missing>           6 weeks ago         /bin/sh -c #(nop)  ENV NGINX_VERSION=1.13.3     0B
<missing>           2 months ago        /bin/sh -c #(nop)  MAINTAINER NGINX Docker...   0B
<missing>           2 months ago        /bin/sh -c #(nop)  CMD ["/bin/sh"]              0B
<missing>           2 months ago        /bin/sh -c #(nop) ADD file:9d67752278c0e5a...   3.99MB
```

nginx:alpine

Docker Layers



Tagging Docker Images

- Docker images are identified by a unique identifier (`4ef8f273e964`)
- Tags provide an additional naming construct - format
  hub.tom-adams.net:5000/tca/nginx:1



- An image can have several tags

Workshop uses two pronged versioning approach

- Version number as part of the tag
  - Based on Jenkins build number
  - Simple sequential number that can be traced back to a specific build.

- Use container label to contain git describe
  - Same as version number used in application version.
  - Additional data point to verify the version of the artifact.

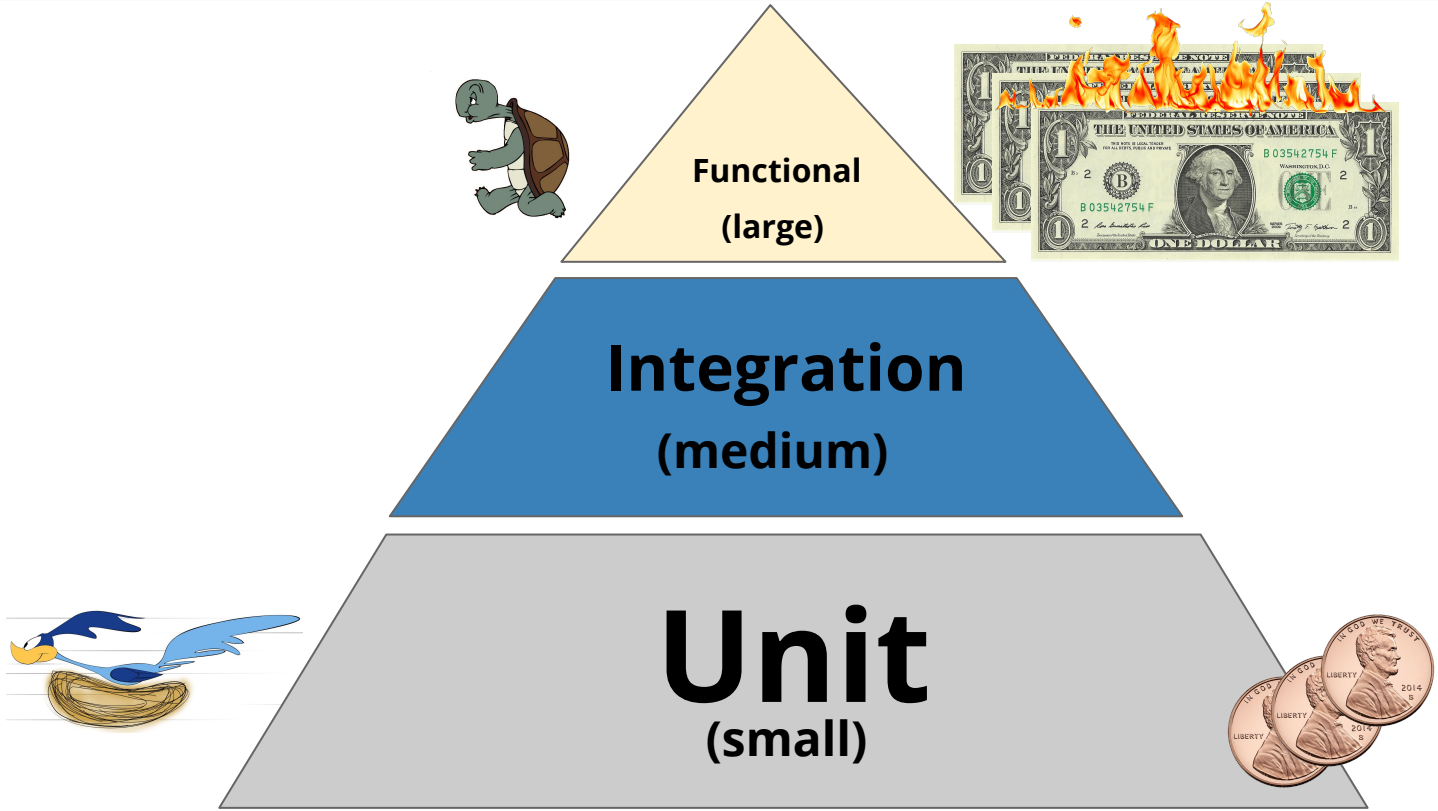
Docker Tag - “latest”

- Default value for an image version, when not specified on the tag command.
- If a version is specified the latest version is not applied to the image.
- Possible to explicitly tag images as latest.

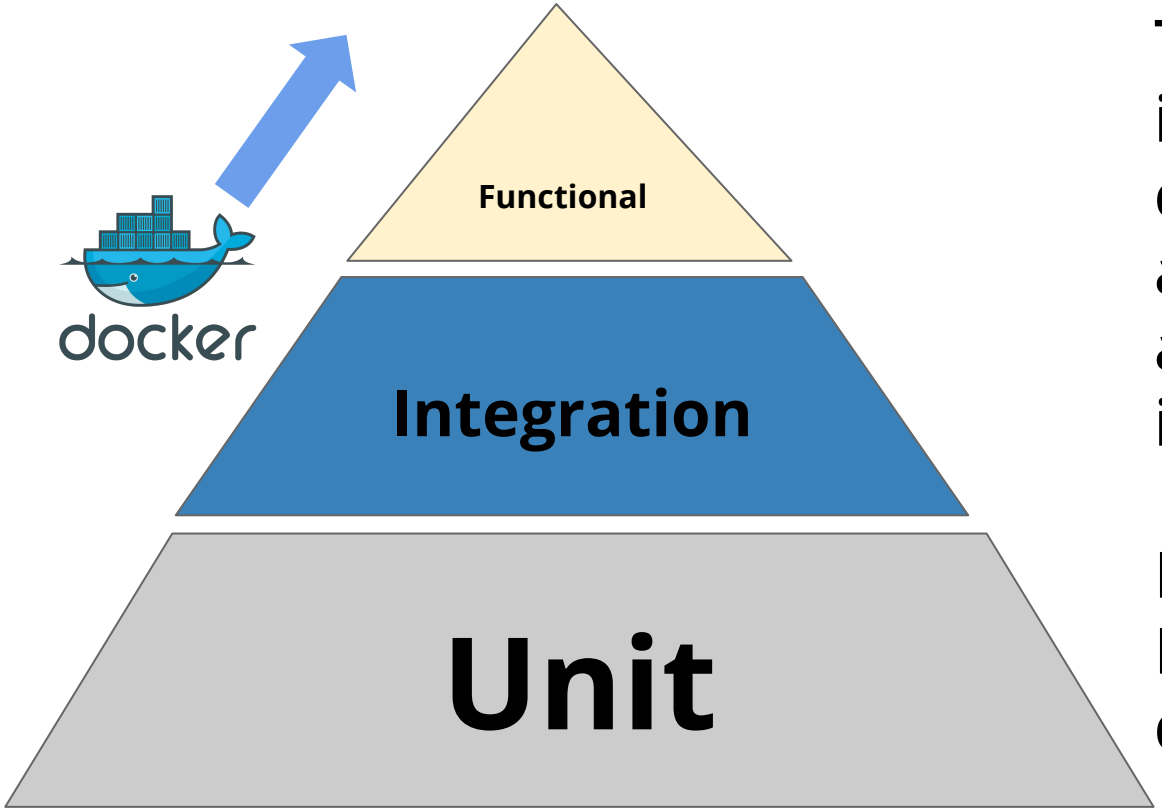
General rule - steer clear of attempting to use the latest tag for image versioning.

If you don’t like testing your product, most likely your customers won’t like testing it either.

Testing pyramid



Testing pyramid with Docker



Treat Docker images as first class build artifacts. Test against Docker images.

Don't simply bolt Docker on during deployment.

Docker Compose

- Tool for defining and running multi-container Docker applications
- Using a docker-compose script file you identify the containers to run and their interdependencies
- Use cases
  - Running complex environments on a development workstation
  - Defining a lower level (small) environment
  - Executing automated functional tests
  - Production multi-node deployments using Docker Swarm
    - Author additional compose configurations and combine yml files
    - Use `docker stack` set of commands

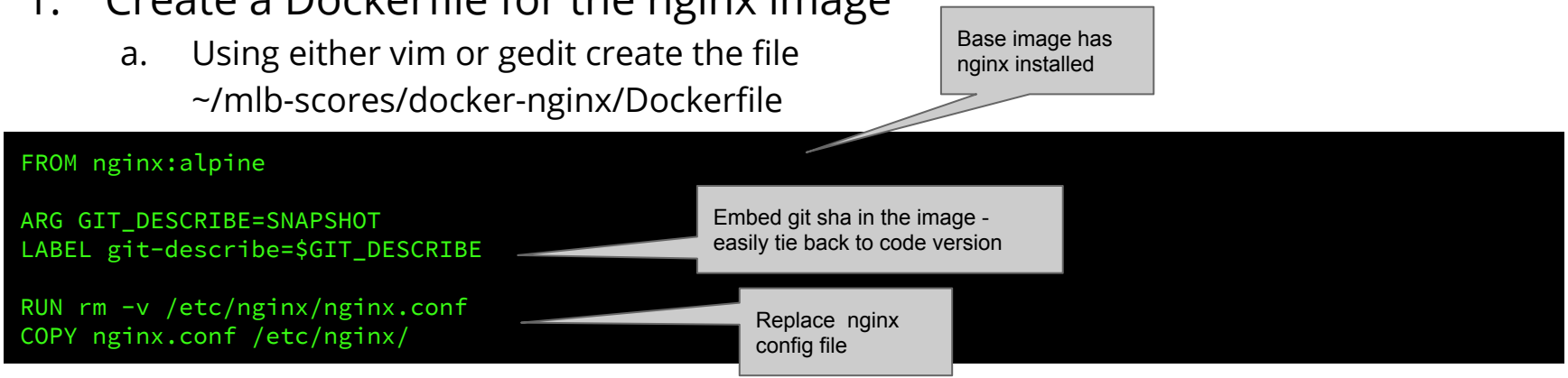
Lab 3 - Creating Docker build artifacts

Goals:

- Define docker files for mlb-scores and nginx
- Test docker image build
- Use docker compose to start the containers together

Steps:

1. Create a Dockerfile for the nginx image
   a. Using either vim or gedit create the file ~/mlb-scores/docker-nginx/Dockerfile



Lab 3 - Creating Docker build artifacts

Steps

2. Create the nginx image using the docker build command

```
$ cd ~/mlb-scores/docker-nginx
$ docker build --tag hub.tom-adams.net:5000/<initials>/nginx:0 \
               --build-arg GIT_DESCRIBE=$(git describe) .
```

Tags the image as version 0

3. Create a Dockerfile for the mlb-scores image
   a. Using either vim or gedit create **~/mlb-scores/docker-mlb-scores/Dockerfile**

```
FROM openjdk:alpine

ARG GIT_DESCRIBE=SNAPSHOT
LABEL git-describe=$GIT_DESCRIBE

RUN mkdir /opt
COPY mlb-scores*.jar /opt/.
COPY startMlb.sh /opt/.

EXPOSE 8080
CMD ["/opt/startMlb.sh"]
```

Base image has Java 8 installed

Copy executable jar into the image

Start container command

Lab 3 - Creating Docker build artifacts

Steps

4. Copy the downloaded jar in the docker-mlb-scores directory

```
$ cp ~/Downloads/mlb-scores-1-.?-???.jar ~/mlb-scores/docker-mlb-scores/.
```

5. Create the mlb-scores image using the docker build command

```
$ cd ~/mlb-scores/docker-mlb-scores
$ docker build --tag hub.tom-adams.net:5000/<initials>/mlb-scores:0 \
               --build-arg GIT_DESCRIBE=$(git describe) .
```

Lab 3 - Creating Docker build artifacts

Steps

Yaml files have strict indentation rules - use two spaces.

6. In the ~/mlb-scores directory create a docker-compose.yml file

```
version: '2'
services:
  mlb-scores-web:
    image: "hub.tom-adams.net:5000/<initials>/mlb-scores:${TAG_VERSION}"
    container_name: <initials>-mlb-scores_${TAG_VERSION}
  mlb-scores-nginx:
    image: "hub.tom-adams.net:5000/<initials>/nginx:${TAG_VERSION}"
    container_name: <initials>-nginx_${TAG_VERSION}
    links:
      - mlb-scores-web
    ports:
      - "80"
```

TAG_VERSION is an environment variable that references a specific version of the image

Allow nginx container to have network connection to mlb-scores-web

Port 80 will be exposed, can be mapped to a host port

7. Set TAG_VERSION to zero

```
$ export TAG_VERSION=0
```

Lab 3 - Creating Docker build artifacts

Steps

8. Start both containers using docker-compose

```
$ cd ~/mlb-scores
$ docker-compose up
. . .
<standard out for both containers>
```

9. In a new terminal tab: check to see how nginx port 80 is mapped

```
$ docker-compose ps
      Name                Command           State           Ports
-------------------------------------------------------------------------
tca-mlb-scores_11   /opt/startMlb.sh        Up      8080/tcp
tca-nginx_11        nginx -g daemon off;    Up      0.0.0.0:32776->80/tcp
```

Port 80 is mapped to 32773
(yours will be different)

Lab 3 - Creating Docker build artifacts

Steps

10. Verify container label value

```
$ docker inspect tca-nginx_11

[
    {
        "Id": "a31f21be8442b63e271ccc99cf28f67c11d20934df94974616d8caea5e82456a",
...
            "OnBuild": null,
            "Labels": {
...
                "com.docker.compose.service": "mlb-scores-web",
                "com.docker.compose.version": "1.8.0",
                "git-describe": "1-21-gb178677"
            }
        },
        "NetworkSettings": {
...
```

Reference your container name from docker-compose ps output

Should match your local git describe

Lab 3 - Creating Docker build artifacts

Steps

11. Test site using Firefox

```
$ firefox http://localhost:<port>/mlb-scores/scores
```

Using a random port helps to mitigate port conflicts on testing host machines.

12. Test site using local functional test suite

```
$ cd ~/mlb-scores
$ gradle localFunctionalTest --rerun-tasks
```

a. Did the test pass (no it doesn't, but why)?
b. Review the ~/mlb-scores/build.gradle script tasks for localFunctionTest and functionalTest.
c. If you run the gradle functionalTest does it pass? why?

Lab 3 - Creating Docker build artifacts

Steps

13. Commit new files

```
$ git add .
$ git commit -m"Adding docker definition files"
```

    a. Ensure mlb-scores_Build job is green.

Lab 3 - 3:25pm

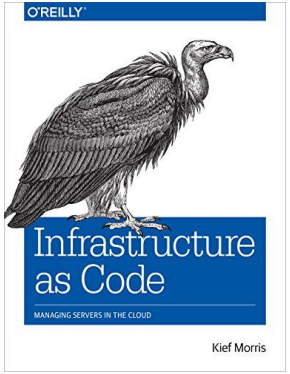




Why use containers as build artifacts?

We are moving from the “Iron age” to the “**Container Age**”

- Iron Age
  - Software directly bound to hardware
  - Long lived (snowflake) servers
  - Manual configuration driven by lots of up front planning



- Container Age
  - Software decoupled from the hardware
  - Short lived (phoenix) servers
  - Automated configuration driven by an explosion in the number of servers

Why use containers as build artifacts?

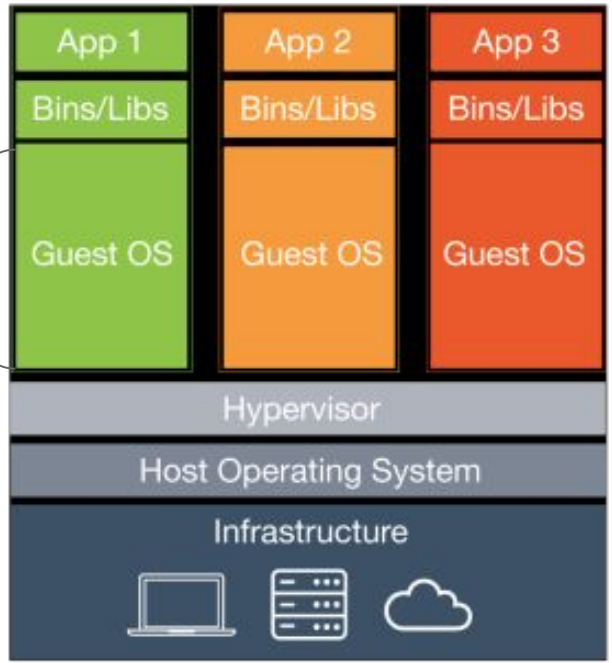
- Docker images contain both code and OS level configuration.
  - Immutable (disposable) infrastructure.
  - Decouple application from specific server
- Test code and configuration together.
  - Easier to create production like environments.
  - Environments can be quickly created and destroyed

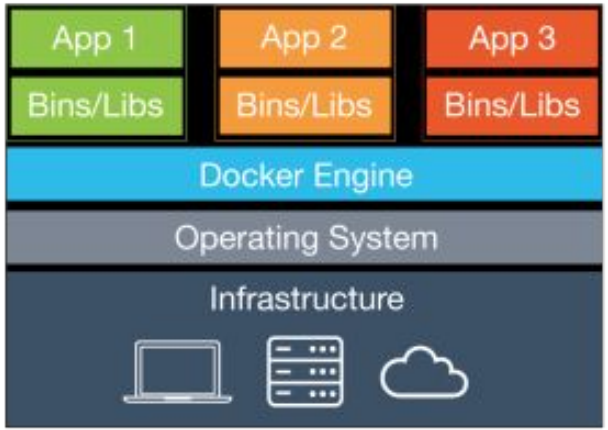
- Cost savings
  - Efficiently utilize server resources.
  - Simple scaling model

Virtual Machine vs. Containers

VM OS instances require CPU/RAM

Abstraction of physical hardware





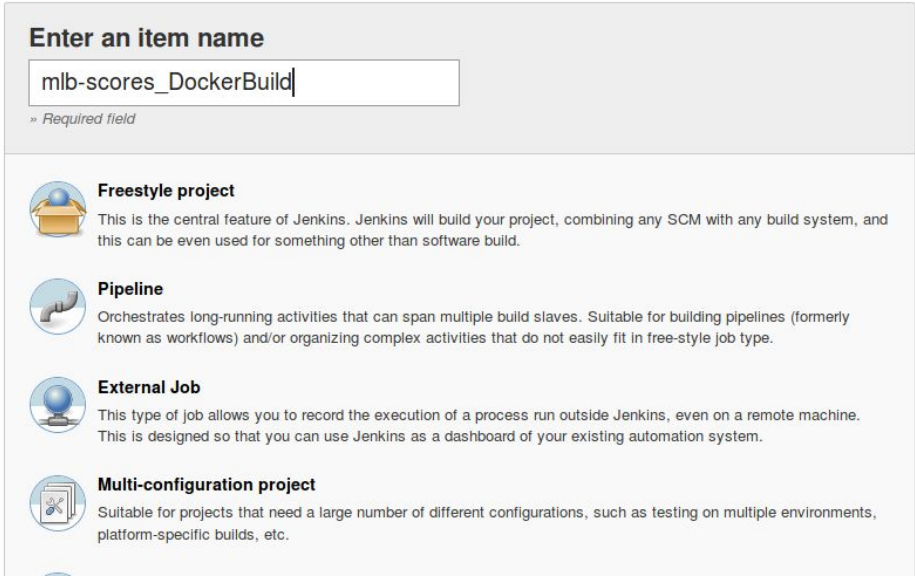
Guest OS not present

Abstraction of application layer

Lab 4 - Moving docker builds to Jenkins

Steps

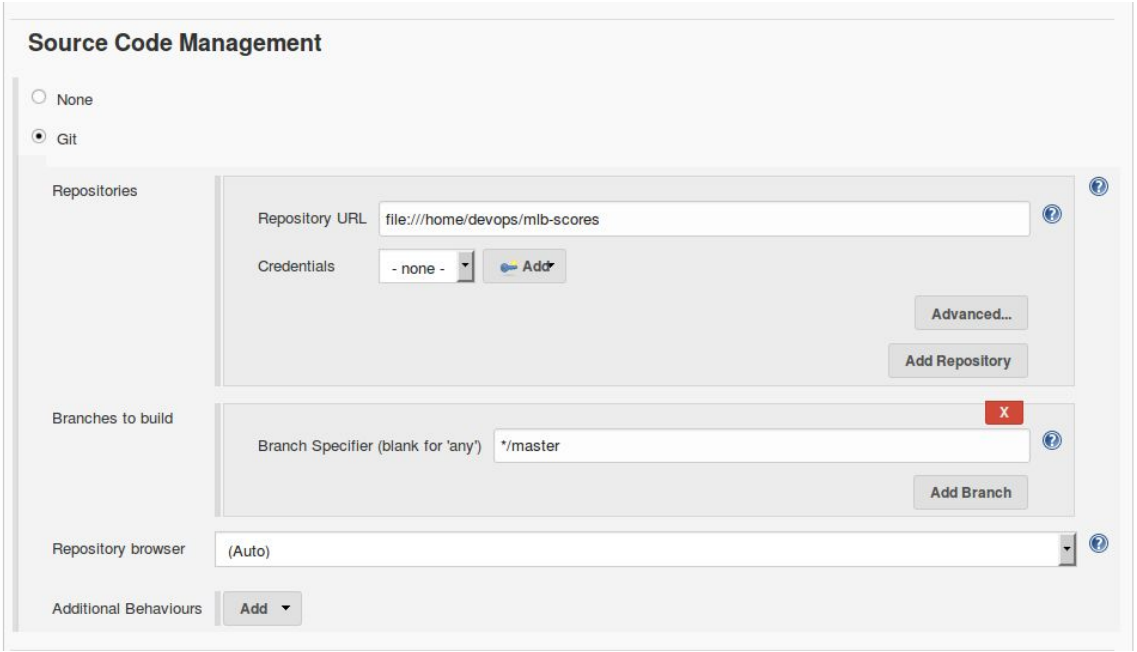
1. Define a mlb-scores_DockerBuild job in Jenkins
    a. From the Jenkins home page (localhost:9090) click on “New Item”
    b. Enter the new job name and click Freestyle project and Ok button



Lab 4 - Moving docker builds to Jenkins

Steps

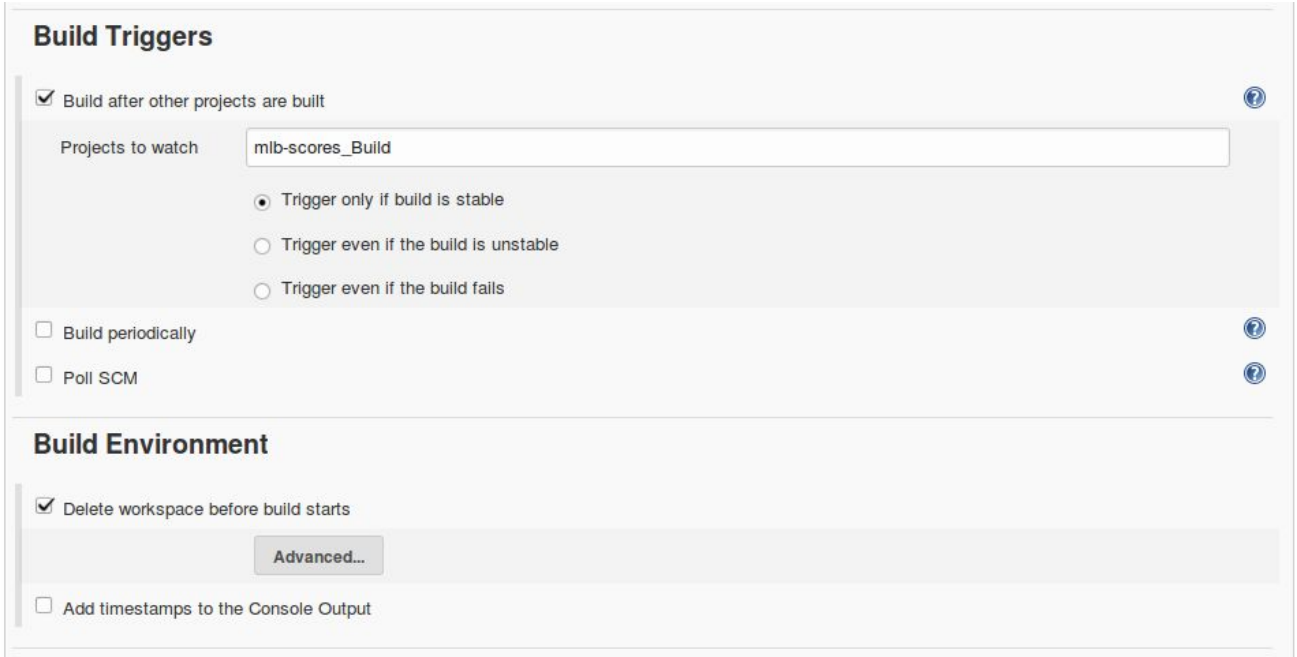
2. On the config page enter the Source Code Management information
   `file:///home/devops/mlb-scores`



Lab 4 - Moving docker builds to Jenkins

Steps

3. Enter the Build Triggers and Build Environment options



Lab 4 - Moving docker builds to Jenkins

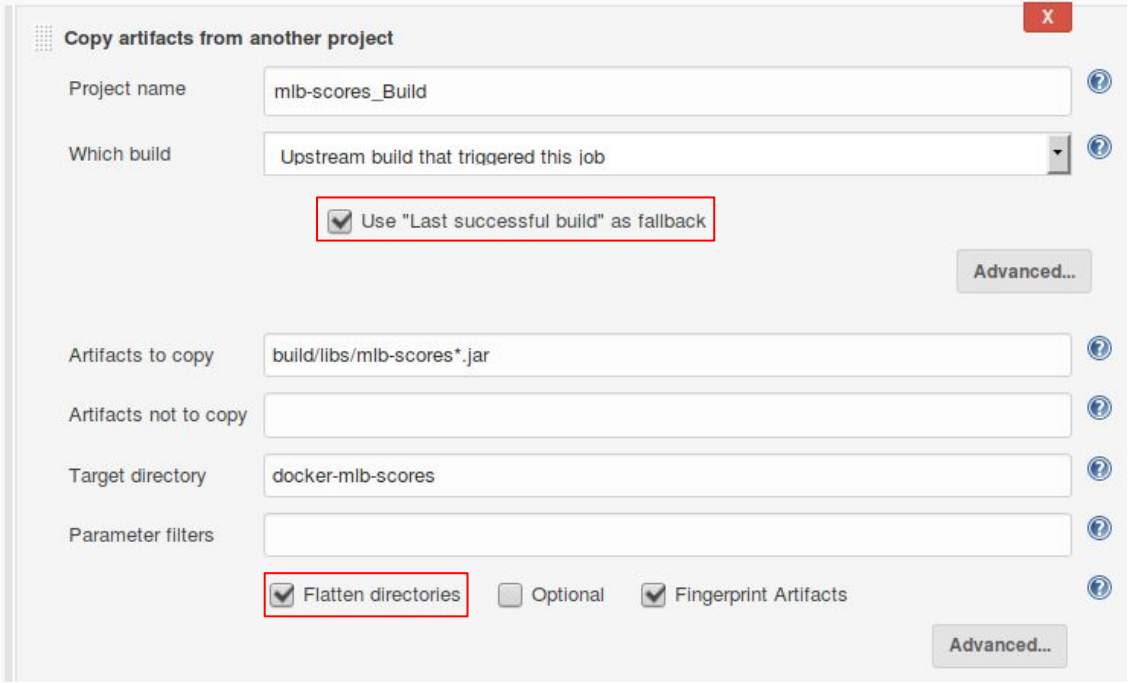
Steps

4. Create five build steps for the job (details on next slides)
    a. Copy the jar file from the triggering build to the docker-mlb-scores directory
    b. Docker build the nginx image
    c. Docker build the mlb-scores-web image
    d. Execute the functional tests
    e. Push docker image to docker registry (hub.tom-adams.net:5000)

Lab 4 - Moving docker builds to Jenkins

Steps

5. Copy jar from triggering job



Lab 4 - Moving docker builds to Jenkins

Steps

6. Define the docker build script for both images
   a. Create the following file in ~/mlb-scores/scripts/docker-build.sh

```
#!/bin/sh
set -e

docker build --tag ${DOCKER_HUB}/<initials>/nginx:${BUILD_NUMBER} \
             --build-arg GIT_DESCRIBE=$(git describe)      \
                   ${WORKSPACE}/docker-nginx

docker build --tag ${DOCKER_HUB}/<initials>/mlb-scores:${BUILD_NUMBER} \
             --build-arg GIT_DESCRIBE=$(git describe)        \
                  ${WORKSPACE}/docker-mlb-scores
```

Unique name for your image

Find Dockerfile in this directory

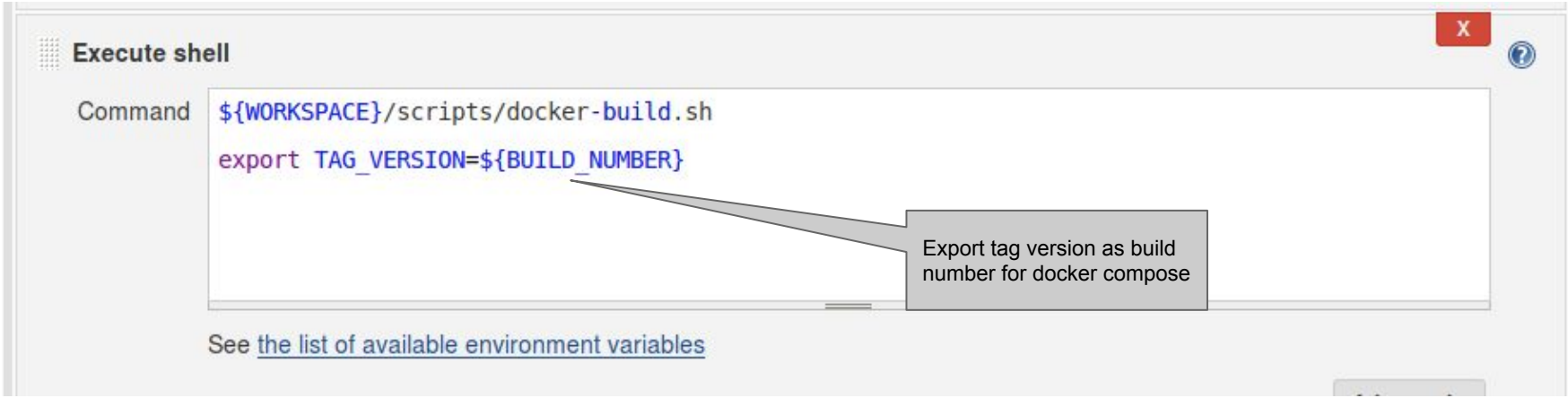
   b. Change permissions to make file executable

```
$chmod +x ~/mlb-scores/scripts/docker-build.sh
```

Lab 4 - Moving docker builds to Jenkins

Steps

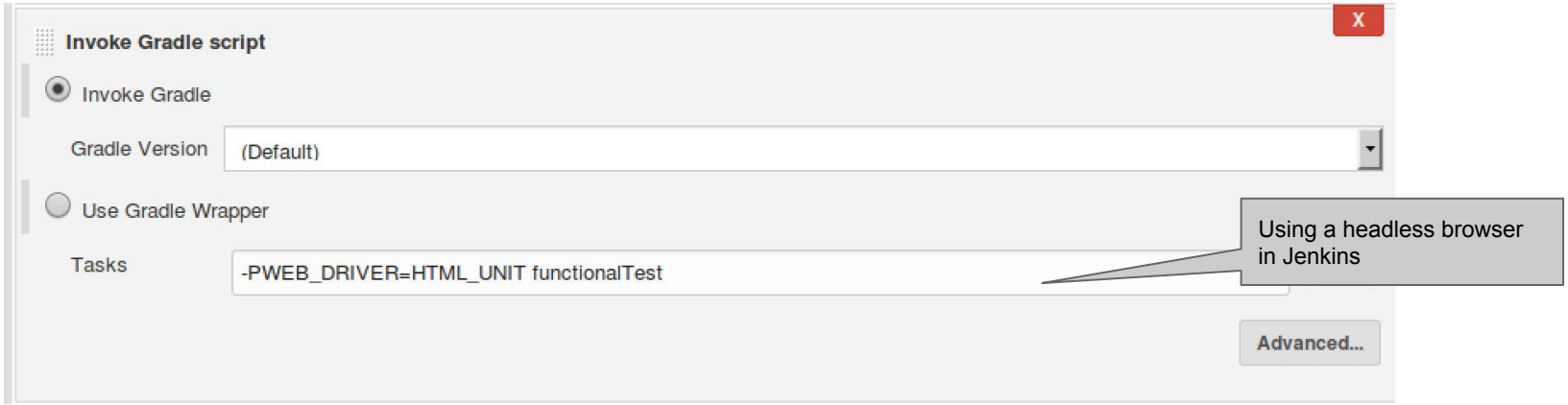
7. Define Jenkins step to build docker images



Lab 4 - Moving docker builds to Jenkins

Steps

8. Define functional test step, and click Save button



Lab 4 - Moving docker builds to Jenkins

Steps

9. Define the docker push script for both images
    a. Create the following file in ~/mlb-scores/scripts/docker-push.sh

```
#!/bin/sh
set -e

docker push ${DOCKER_HUB}/<initials>/nginx:${BUILD_NUMBER}
docker push ${DOCKER_HUB}/<initials>/mlb-scores:${BUILD_NUMBER}
```

Unique name for your image

    b. Change permissions to make file executable

```
$chmod +x ~/mlb-scores/scripts/docker-push.sh
```

Lab 4 - Moving docker builds to Jenkins

Steps

10. Define Jenkins step to push docker images



11. Save mlb-scores_DockerBuild

Lab 4 - Moving docker builds to Jenkins

Steps

12. Commit docker scripts and ensure build is successful

```
$git add .

$git commit -m”Adding docker scripts”
Scheduled polling of mlb-scores_Build
No Git consumers using SCM API plugin for: file:///home/devops/mlb-scores
[master 6df41ef] adding docker scripts
 2 files changed, 12 insertions(+)
 create mode 100755 scripts/docker-build.sh
 create mode 100644 scripts/docker-push.sh
```

13. Ensure build runs and all steps are green
14. Remove WEB-DRIVER functional test parameter

    `-PWEB_DRIVER=HTML_UNIT functionalTest to functionalTest`
15. Re-run build pipeline, can you tell the functional test failed?

Lab 4 - Moving docker builds to Jenkins

Steps

16. Verify docker image was pushed to hub

```
$ curl http://hub.tom-adams.net:5000/v2/_catalog
{"repositories":["mlb-scores","nginx","tca/mlb-scores","tca/nginx"]}
```

Should see a repository for each of your two images mlb-scores and nginx

17. Verify docker image has correct version

```
$ curl http://hub.tom-adams.net:5000/v2/tca/mlb-scores/tags/list
{"name":"tca/mlb-scores","tags":["4"]}

$ curl http://hub.tom-adams.net:5000/v2/tca/nginx/tags/list
{"name":"tca/nginx","tags":["4"]}
```

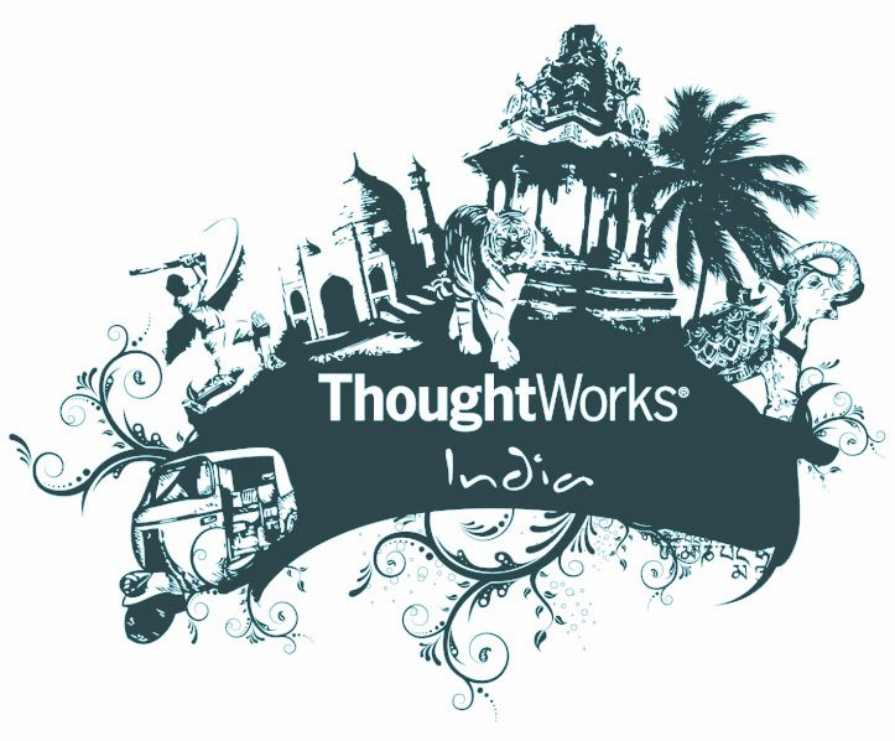
Lab 4 - Questions

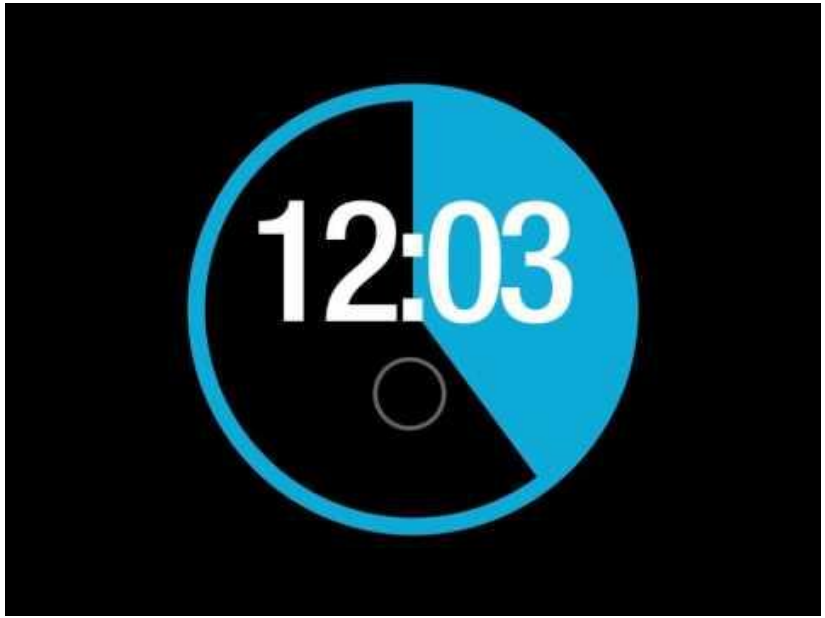
- Why did we create docker build scripts? Why not simply put these commands into a Jenkins shell script step?

- How are the environment variables used in the docker registry push script configured - what are their values?

- How is the TAG_VERSION in docker-compose.yml set for the functional tests?

Lab 4 - 4:00pm



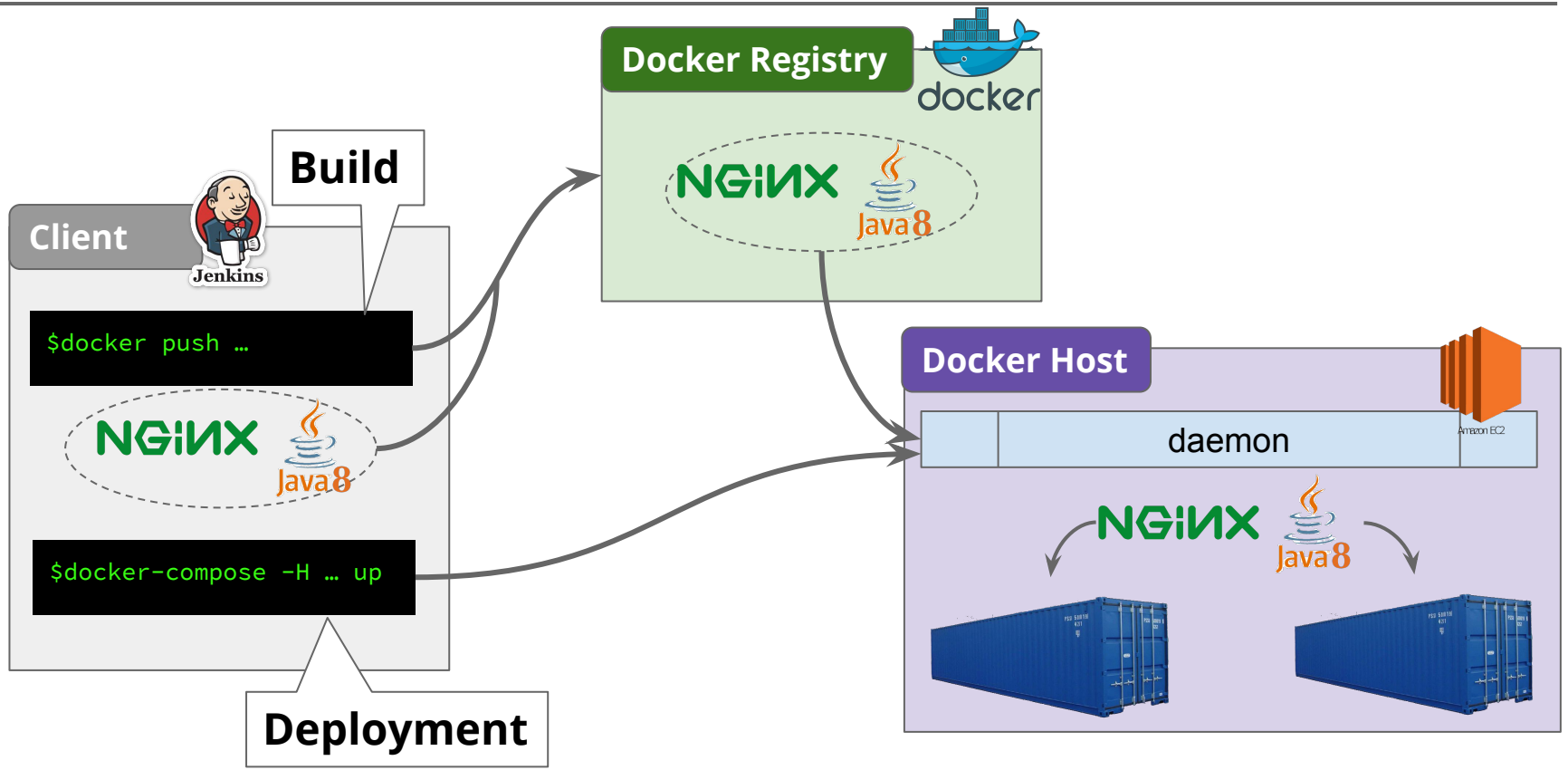


Deploying Docker Containers

- Docker container “orchestration” is a term used to define a set of tools that deploy containers to multi-node clusters
  - Amazon ECS
  - Kubernetes
  - Docker Swarm
- Typical deployment is done from command line using remote APIs and a Docker registry
- Lab uses a remote single-node Docker host
- Running docker commands on remote host

```
$docker -H <hostname>
```

Remote Docker Hosts



Lab 5 - Deploying docker image

Goals

- Create a mlb-scores_Deploy job in Jenkins

  Define two job parameters:

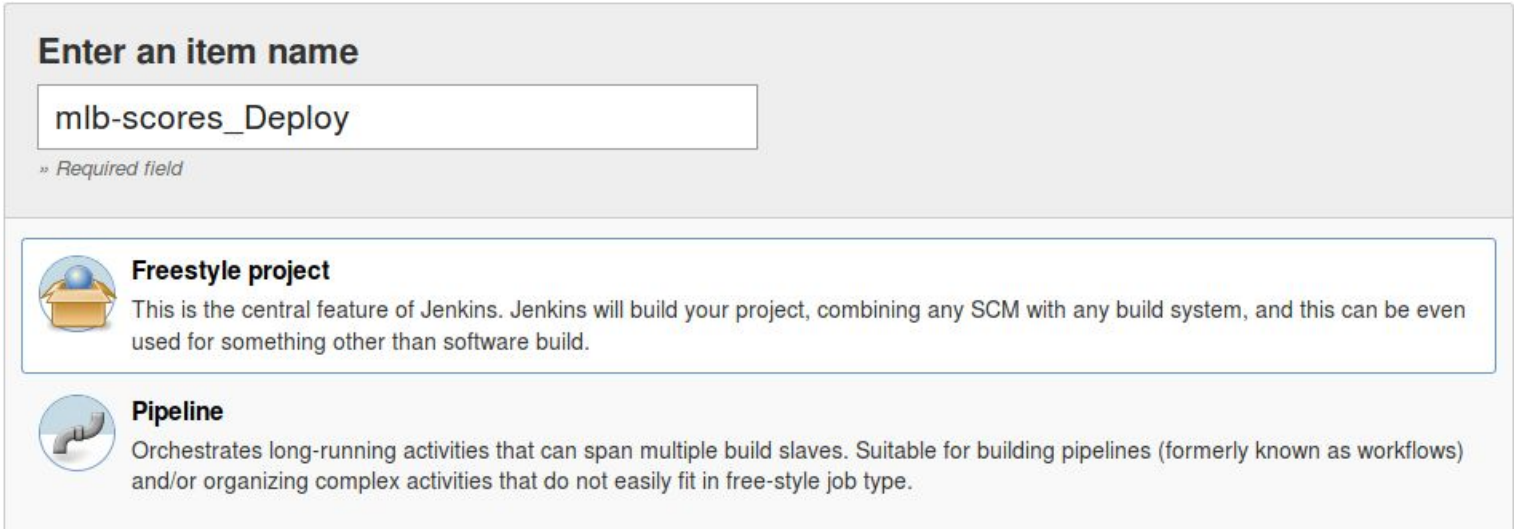
  a. Deploy target is choice parameter selected - dev, qa, or prod
  b. Job will deploy a specific version of the docker images using Compose. Version is based on list of successful DockerBuild job numbers.

- Run deploy job and verify containers are executing in development environment.

Lab 5 - Deploying docker image

Steps

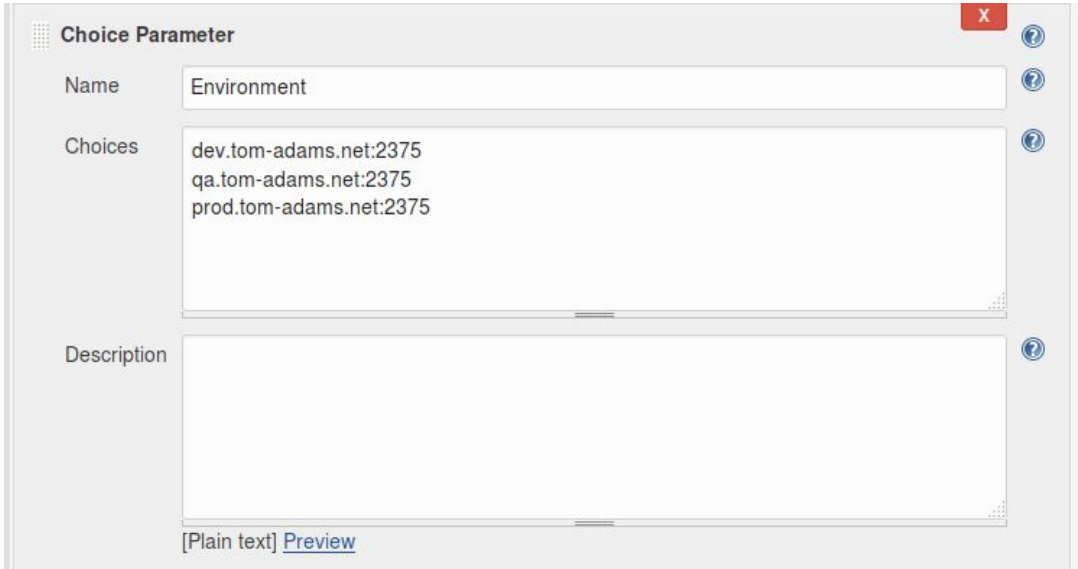
1. Define a mlb-scores_Deploy job in Jenkins
   a. From the Jenkins home page (localhost:9090) click on “New Item”
   b. Enter the new job name and click Freestyle project and Ok button



Lab 5 - Deploying docker image

Steps

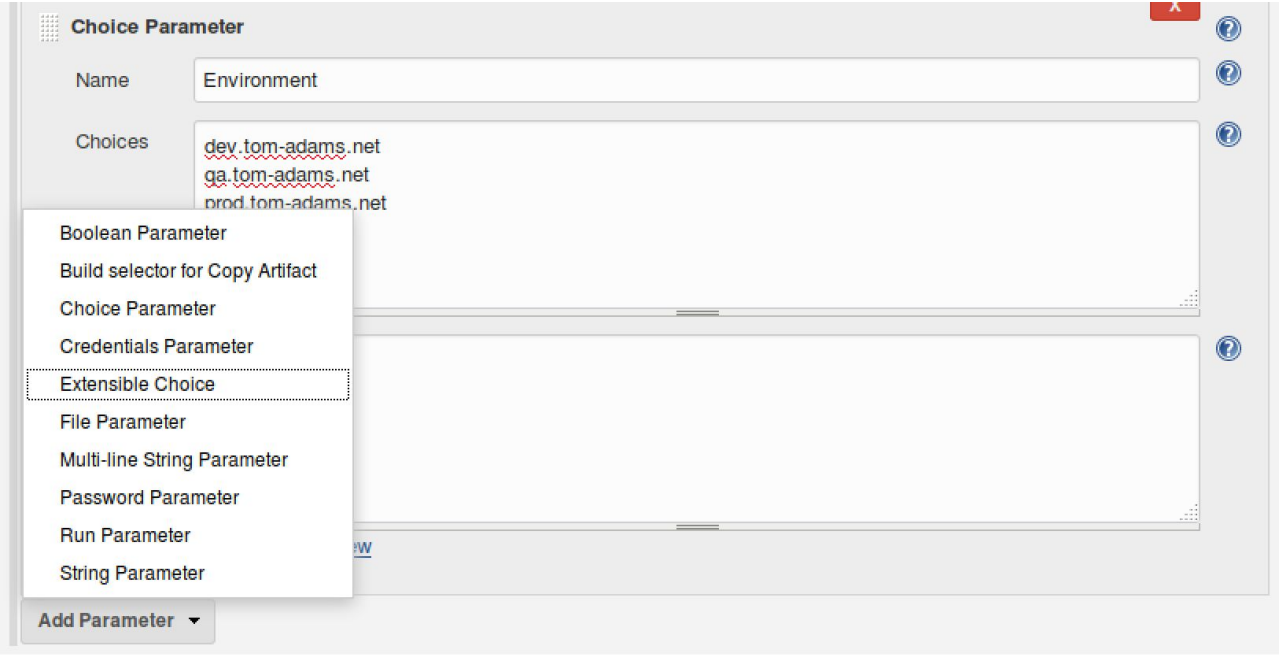
2. Select the checkbox “This project is parameterized” and define the Environment parameter as a “Choice Parameter”



Lab 5 - Deploying docker image

Steps

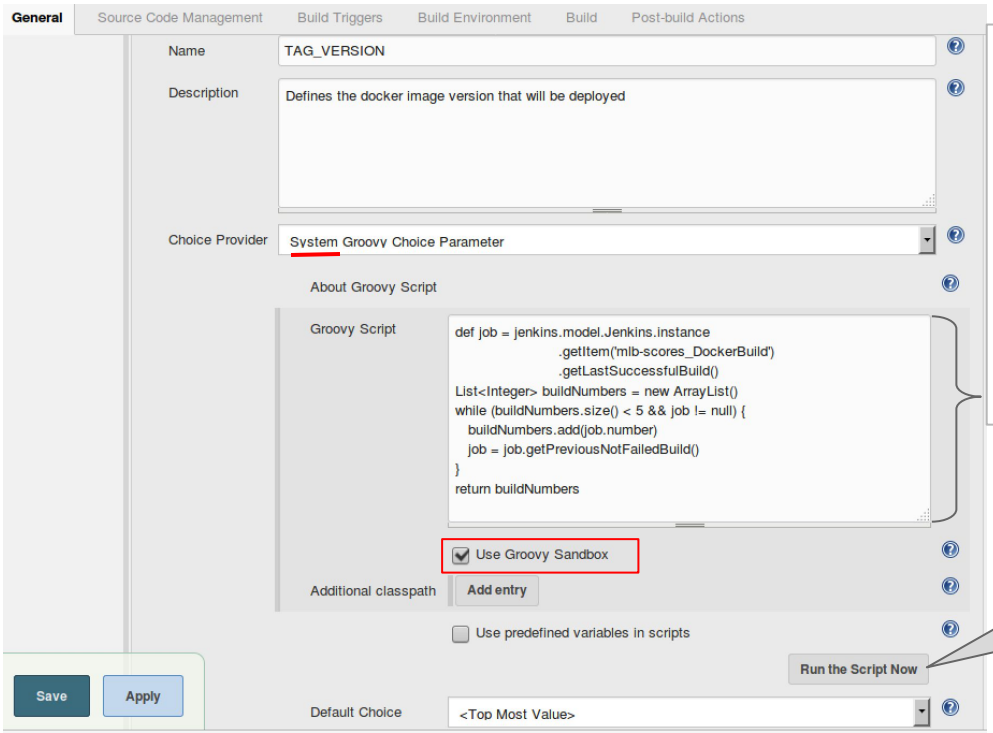
3. Define a second parameter as a “Extensible Choice” - Version to deploy



Lab 5 - Deploying docker image

Steps

4. Use groovy choice parameter with script




```
def job = jenkins.model.Jenkins.instance
                .getItem('mlb-scores_DockerBuild')
                .getLastSuccessfulBuild()
List<Integer> buildNumbers = new ArrayList()

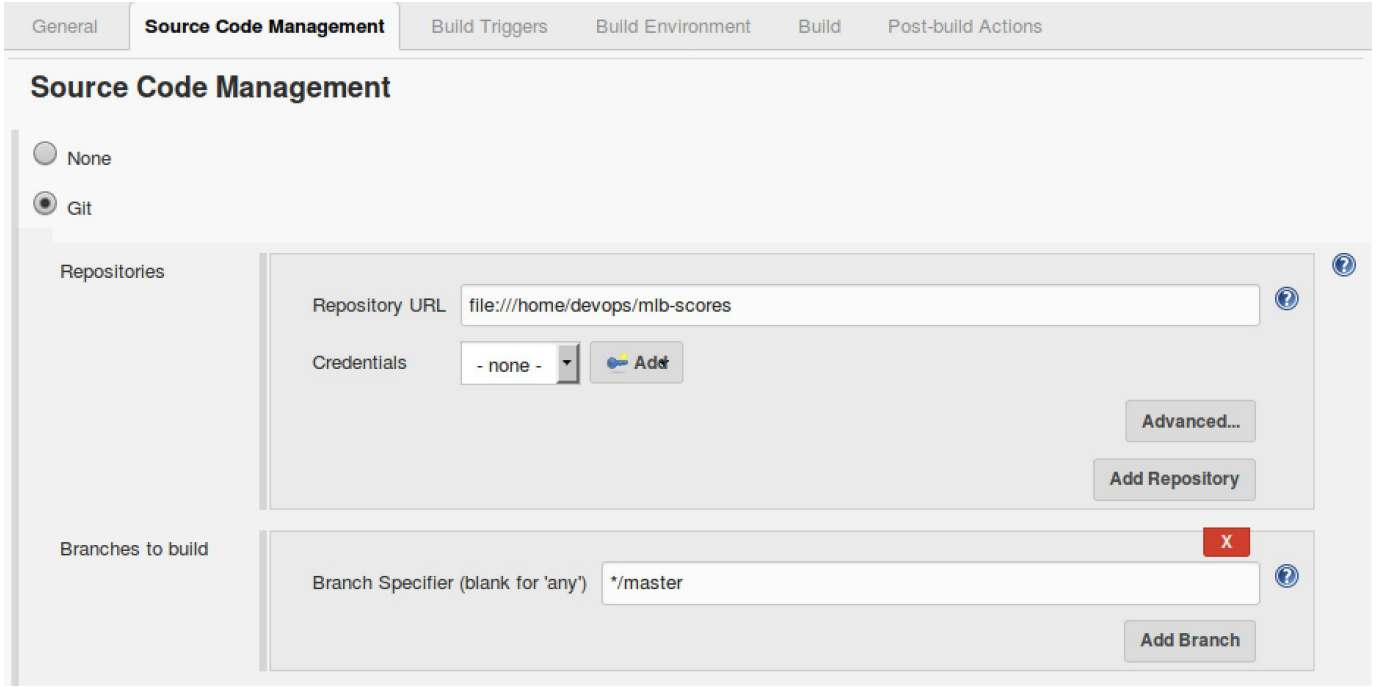
while (buildNumbers.size() < 5 && job != null) {
   buildNumbers.add(job.number)
   job = job.getPreviousNotFailedBuild()
}
return buildNumbers
```

After you paste in the script click the “Use Groovy Sandbox” checkbox and then click Run the Script Now - you should see a list of successful DockBuild numbers

Lab 5 - Deploying docker image

Steps

5. Define git repository



Lab 5 - Deploying docker image

Steps

6. Define the docker deploy script
   a. Create the following file in ~/mlb-scores/scripts/docker-deploy.sh

```
#!/bin/sh
set -e

docker-compose -p <initials>-scores -H ${Environment} up -d
```

-p is the project parameter, allows everyone's docker containers to run together.

   b. Change permissions to make file executable

```
$chmod +x ~/mlb-scores/scripts/docker-deploy.sh
```

   c. Commit new file in git

```
$git add .

$git commit -m"adding docker deploy script"
```

Lab 5 - Deploying docker image

Steps

7. Add shell deploy step to Jenkins

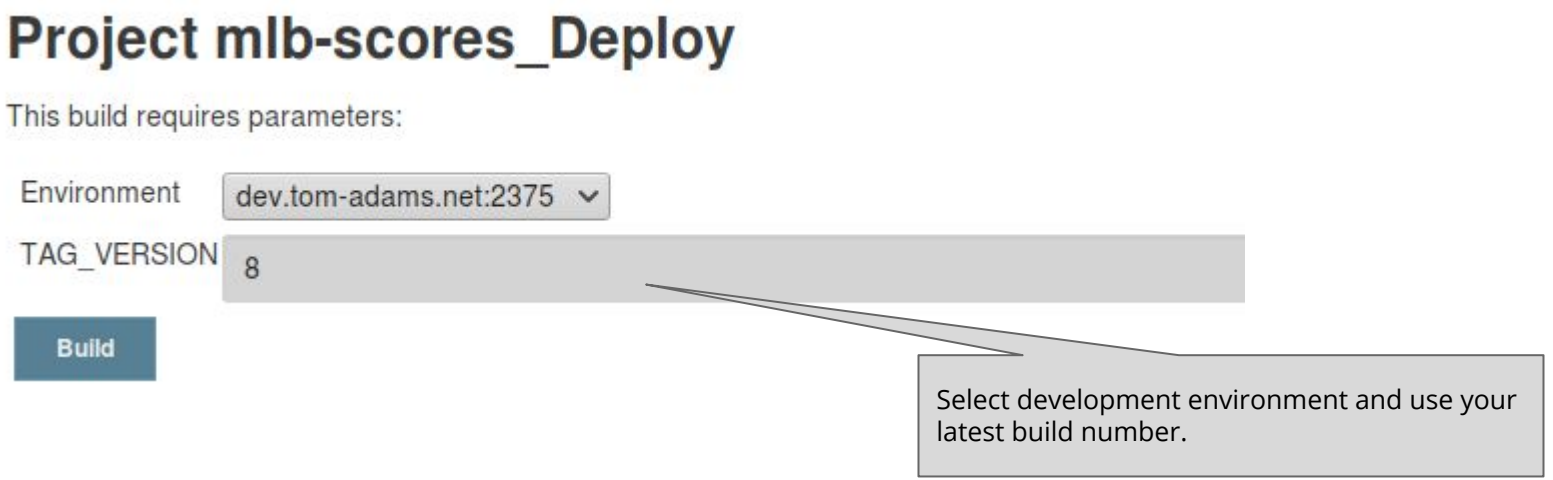


8. Save job

Lab 5 - Deploying docker image

Steps

9. Run Deployment job



Lab 5 - Deploying docker image

Steps

10. Run mlb-scores_Deploy job
11. Verify Docker containers are running on server

```
$ docker -H dev.tom-adams.net:2375 ps

CONTAINER ID        IMAGE                                      COMMAND                  CREATED             STATUS              PORTS
NAMES
d9abdf2b55fa        hub.tom-adams.net:5000/tca/nginx:4         "nginx -g 'daemon ..."   42 seconds ago      Up 40 seconds
0.0.0.0:32770->80/tcp    mlbscoresdeploy_mlb-scores-nginx_1
59f7ab54b1e9        hub.tom-adams.net:5000/tca/mlb-scores:4    "/opt/startMlb.sh"       44 seconds ago      Up 42 seconds
8080/tcp                 mlbscoresdeploy_mlb-scores-web_1
```

12. Open page in browser to validate containers running

```
$ firefox http://dev.tom-adams.net:32770/mlb-scores/scores
```

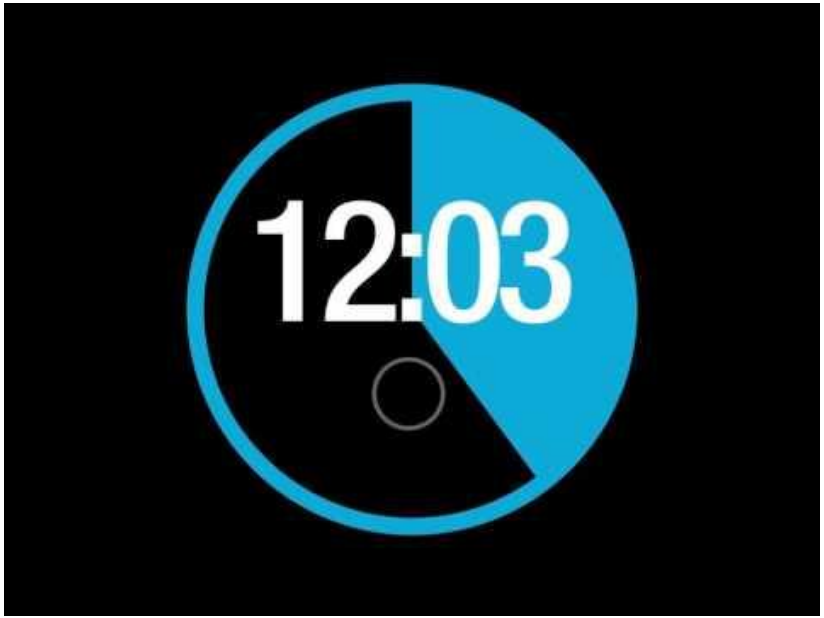
Lab 5 - Extra Credit

- Add a step to the deploy job to verify the containers started.

- Update verify step to ensure the correct ***version*** of the containers are running.

Lab 5 - 4:45pm





Remote Docker Server / Hub

- The example used in this workshop was not secure.
- OK for workshop, technical spikes maybe some lower environments.
- Production environments should be secure using shared certificate authorities, TLS etc.
- Single node docker deployment targets are only good for workshops (not real environments)



Workshop Take Aways

- By packaging more of the software stack together Docker containers improve the testability of build pipeline artifacts.
  - Requires Docker to be a first class build artifact
  - Use docker images in all lower environments for automated and manual functional tests
- As with all build artifacts - they must be versioned and easily tie back to a specific commit.
- All code used in a build pipeline needs to be kept in version control.
- The build server process should be able to run easily on local workstation.
- Still need to continuously integrate code into a single development trunk daily (Docker doesn’t make that any easier).

Next Steps?

- Continue learning using the VM and sample application.
  - Research and experiment with different docker image commands
  - What are other docker and docker-compose commands not used in the workshop
- Expand the simple containers defined in the workshop.
  - Define a user within the docker image to run the applications
  - Create two mlb-scores applications
- What would need to change to deploy these images to docker swarm?
- What about using Docker containers to perform builds / compiles?
  - Great solution, especially for tool chains that require heavy OS level dependencies (ruby, python, npm) and shared build slaves.
- Change build to use pipeline Jenkins job definitions

GO BE AWESOME

For questions, suggestions or feedback:

Tom Adams

tadams@thoughtworks.com

ThoughtWorks®