Docker: A boon for the modern developer

Make the transition to containerized apps to increase coding productivity and boost your development IQ

Sing Li

January 05, 2016

Use Docker to code more productively and expand your skills more efficiently. Declutter your development environment, eliminate dependency conflicts, and reduce development and learning time by working with containerized applications.

The role of Docker as a sort of Swiss Army knife for DevOps is well documented. But Docker-managed application containers are useful for more than deploying servers in the cloud. Docker containers can also aid in development and increase productivity dramatically in many common development scenarios. This tutorial focuses on how Docker can be useful from a developer's perspective. I introduce Docker, explain basic concepts and terminology, and present a series of hands-on development examples.

You'll see that:

- Docker can facilitate Node.js web application coding.
- You can quickly create and test Java™ Enterprise Edition (Java EE) applications on Apache Tomcat server by using Docker.
- Docker can expedite coding, testing, and deployment of Python web apps via the Bottle framework.
- You can create and test an entire three-tier Node.js application with a NoSQL database back end in minutes by using a Docker container.

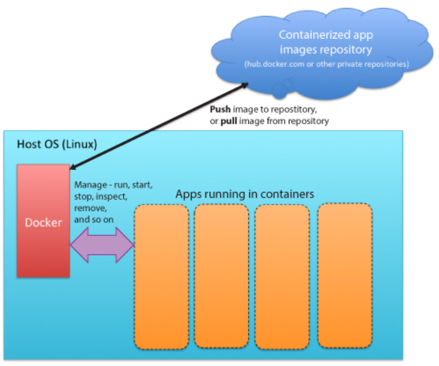
Docker, golang, and Bluemix

Read the companion tutorial "Containerize golang apps with Docker and Bluemix" to extend your polyglot exploration to the Go Programming Language. And discover how easy it is to deploy a Dockerized three-tier app to Bluemix™ via IBM Containers for Bluemix.

You can garner all of these benefits on a single Linux system, without disturbing any preset configurations. After working through the examples, you'll feel right at home running various Docker containers to increase productivity in your everyday development workflow.

Docker basics

Docker is a container manager. Containers bundle an application with its dependencies. You instantiate each container from an image stored in a repository and run containers in isolated virtualized environments within the host OS. Because the virtualization technology is (typically) lightweight, you can run multiple containers simultaneously:


Containerized app images repository
(hub.docker.com or other private repositories)
Push image to repostitory, or **pull** image from repository
Host OS (Linux)
Docker
Manage - run, start, stop, inspect, remove, and so on
Apps running in containers

Grasping essential terminology

It's important to understand the difference between an image and a container. The two concepts are closely related, and the distinction is often confusing to beginners.

An *image* is a static set of layers. No runtime behaviors are associated with an image. Images are stored in repositories (Docker Hub, for example).

A *container* is a running instance started from an image. Although a container can be stopped — with its states temporarily frozen on disk — it's still a container.

The potential confusion stems from the fact that by committing and saving a container, you can turn the container into an image. In this case, you can think of the resulting image as a "freeze-dried" version of the running container.

You can use the Docker container manager to:

- Run containers from images
- Inspect and manipulate properties of running containers
- Stop running containers
- Execute additional tasks within running containers
- Start or restart running containers
- Perform many other management tasks

Containerized apps can run almost anywhere: on a desktop PC, on a server, in a public cloud, or even on some mobile phones.

Linux is by far the most widely used and supported Docker platform. The containerization (lightweight virtualization) technologies managed by Docker are most mature on Linux platforms, using recent Linux features such as Control Groups and namespaces.

How to run Docker

You run the Docker container manager via the `docker` command-line client, which has many available switches and options. Here's a simplified command to run a containerized app image (for the MySQL database in this case):

```
docker run -d mysql:5.5
```

The Docker project operates the publicly accessible Docker Hub. Users can sign up and create their own containerized app images repository, then push images for shared public consumption. For example, you can find images for Tomcat server, Node.js, and most popular open source databases. Docker Hub operates in the same spirit as GitHub, in that app images are shared and collaboratively created by the global DevOps and developer community.

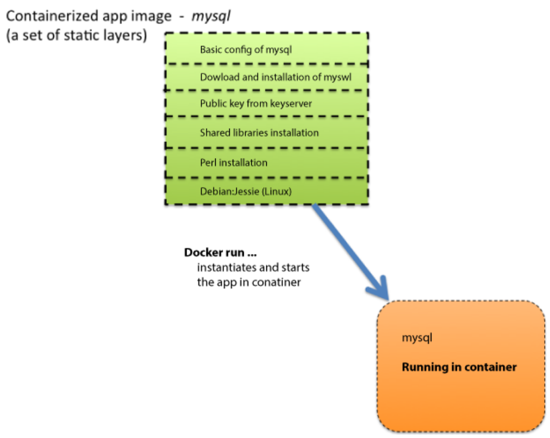
When you run the `docker run -d mysql:5.5` command, for example, the `mysql` (release 5.5) containerized app image is automatically pulled (downloaded) from Docker Hub if you don't already have it stored locally.

Building containerized app images

You can also use the `docker` command-line client to build containerized app images. One way is to build images from a `Dockerfile`— a text file containing instructions on how to install, set up, and configure an application and its dependencies.

Another way to build containerized images with the `docker` command-line client is to handcraft them interactively. You manually install and configure an app and its dependencies inside a running container, and then commit and save the container as an image.

A Docker image consists of a set of layers, with each layer roughly equivalent to the changes written to disk during the installation of a set of applications. Docker manages these layers and enables efficient storage and reuse when images are added or removed. The layers in a `mysql` image, for example, could consist of the Linux OS, Perl, shared libraries, a MySQL installation, and a basic MySQL configuration:



Whether you build a containerized image from a `Dockerfile` or handcraft a custom container, you typically don't start from scratch. Instead, you base your work on an existing `Dockerfile` or an image from Docker Hub. This way, developers and operations can build on each other's work and collaborate in the creation and management of a set of useful images.

The command to send an image to an image repository (such as Docker Hub) is `docker push`. The command to download an image for local use is `docker pull`.

Before you begin

“ Even though you'll be developing with Java, Tomcat, Node.js, Python, and CouchDB, you don't need to install any of them. Such is the beauty of using Docker in your development workflow. ”

Before you begin, you need a Linux system with Docker installed and running on it. To install Docker, follow the installation instructions for your Linux distribution.

If you have no direct access to a Linux system, you can turn to the cloud. Most cloud hosts offer ready-to-run virtual private servers or virtual machines that you can provision with Linux and connect to (via SSH or a web-based terminal) within minutes. These hosts are typically compatible with Docker.

The examples in this tutorial assume that you're running Docker on a Linux system. If you're already running Docker on Mac OS X or another non-Linux system, your containers will likely be running inside an additional hypervisor-virtualized virtual machine (often VirtualBox). In that case, you'll need to modify the commands in the examples accordingly to follow along.

Check your Docker version

Check your Docker version by running:

```
docker --version
```

Efficient layer management

When you pull the images, notice messages that indicate that a layer already exists. Docker manages changes written to disk during image creation via layers, which enables the image changesets to be managed efficiently. For example, the base Ubuntu image, or the installation of a core utility, can be shared by multiple images — saving significant disk space and download time.

If you're running version 1.8.3 or higher, your system is all set now. Even though you'll be developing with Java, Tomcat, Node.js, Python, and Apache CouchDB, you don't need to install or uninstall any of them. Such is the beauty of using Docker in your development workflow.

Pull the images

The first time that you `pull` an image from Docker Hub, you must download it to your local PC's repository. The download can be time consuming, depending on your Internet access speed, but subsequent use of the image will be lightning-fast.

Run the following commands to pull down all of the images that you'll use in the examples:

```
docker pull node:0.10.40

docker pull java

docker pull tomcat:8

docker pull webratio/ant

docker pull python:3.5

docker pull frodenas/couchdb
```

For more information about each of these images, visit the Docker Hub pages for `node`, `java`, `tomcat`, `webratio/ant`, `python`, and `frodenas/couchdb`.

With your setup now complete, you're ready to explore using Docker to work with a Node.js application.

Quickly set up and develop server-side JavaScript code

The javascript directory in the code distribution (see Download) contains the source code for a web-store example application written in Node.js. The app (originally designed and written by Lauren Schaefer) uses the Express web framework with the Jade Node template engine. (For more information about this application, see "Bluemix fundamentals: Deploy a sample Node.js application to the cloud.") By using Docker, you can work with this application on your computer without installing Node.js or any of its dependencies — or needing to troubleshoot package-installation conflicts.

The main program is in app.js. CSS and associated JavaScript files are in javascript/public/static. The Jade templates are in javascript/view.

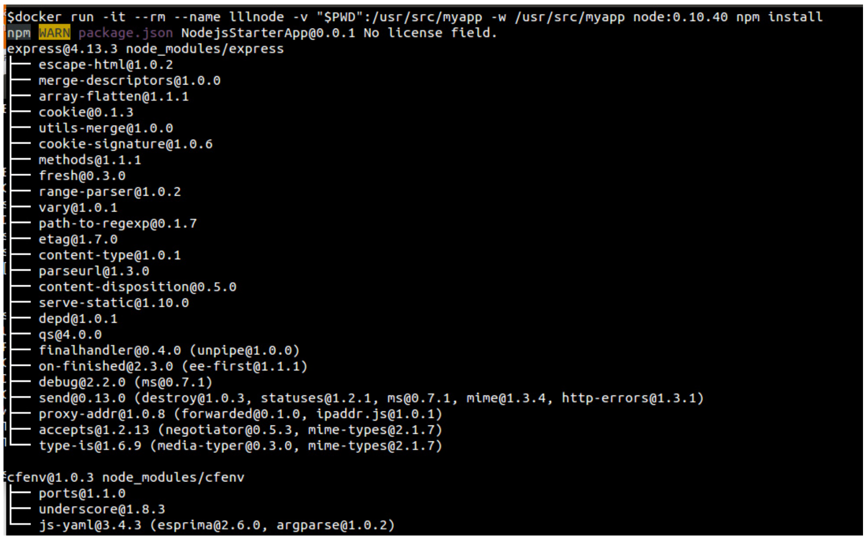
Install the Node.js dependencies for this app by running:

```
docker run -it --rm --name lllnode -v "$PWD":/usr/src/myapp -w /usr/src/myapp node:0.10.40 npm install
```

This command starts the Node container, mounts the current app directory inside the container, and then runs `npm install` on the app:

- The base command is `docker run # node:0.10.40`, which creates a container instance and runs the `node:0.10.40` image that you pulled earlier.
- The `-it` switch specifies that you want a foreground interactive terminal. (The alternative mode is detached background process, specified with `-d`.)
- The `--rm` switch specifies cleanup, meaning that as soon you exit the container, Docker removes it. If you don't specify this switch, the container is persisted to disk in a stopped state, and you can restart it from the point of interruption. However, restarting containers is a frequent source of disk exhaustion for users who forget to remove container instances. If you don't use `--rm`, at any time you can see how many stale stopped containers you have lying around (the number might surprise you) by running:
  ```
  docker ps -a
  ```
- The `--name lllnode` option names the container explicitly, which is useful for referencing the container in other Docker commands. If you don't name the container explicitly, Docker assigns it a generated text name — which is usually less meaningful. You can also reference the container by its internal ID (a long, human-unfriendly string of hex digits).
- The `-v "$PWD":/usr/src/myapp` option creates a volume mount, mounting your current working directory (`$PWD`) inside the container as /usr/src/myapp. The Node.js source code of the app (or any other Node.js source code you might have in the current working directory) can then be accessed inside the container.
- The `-w /usr/src/myapp` option sets the working directory of the command that you're running. In this case, the working directory is changed to the mounted volume.
- The `npm install` command at the end of the `docker run` command is run within the container on the working directory. The net effect is that you're running `npm install` on the current directory through the container — without installing Node.js or any of its prerequisites.

In the command output you can see all of the dependencies that `npm` is installing:



```
$docker run -it --rm --name lllnode -v "$PWD":/usr/src/myapp -w /usr/src/myapp node:0.10.40 npm install
npm WARN package.json NodejsStarterApp@0.0.1 No license field.
express@4.13.3 node_modules/express
├── escape-html@1.0.2
├── merge-descriptors@1.0.0
├── array-flatten@1.1.1
├── cookie@0.1.3
├── utils-merge@1.0.0
├── cookie-signature@1.0.6
├── methods@1.1.1
├── fresh@0.3.0
├── range-parser@1.0.2
├── vary@1.0.1
├── path-to-regexp@0.1.7
├── etag@1.7.0
├── content-type@1.0.1
├── parseurl@1.3.0
├── content-disposition@0.5.0
├── serve-static@1.10.0
├── depd@1.0.1
├── qs@4.0.0
├── finalhandler@0.4.0 (unpipe@1.0.0)
├── on-finished@2.3.0 (ee-first@1.1.1)
├── debug@2.2.0 (ms@0.7.1)
├── send@0.13.0 (destroy@1.0.3, statuses@1.2.1, ms@0.7.1, mime@1.3.4, http-errors@1.3.1)
├── proxy-addr@1.0.8 (forwarded@0.1.0, ipaddr.js@1.0.1)
├── accepts@1.2.13 (negotiator@0.5.3, mime-types@2.1.7)
└── type-is@1.6.9 (media-typer@0.3.0, mime-types@2.1.7)

cfenv@1.0.3 node_modules/cfenv
├── ports@1.1.0
├── underscore@1.8.3
└── js-yaml@3.4.3 (esprima@2.6.0, argparse@1.0.2)
```

After the dependencies are installed, you can run the app through the container. Use this command (type it as a single line):

```
docker run -it --rm --name lllnode --env PORT=8011 --net="host" -v
"$PWD":/usr/src/myapp -w /usr/src/myapp node:0.10.40 node app.js
```

This command is similar to the one you ran previously. But note that the command to run inside the container is now `node app.js`, which starts the app.

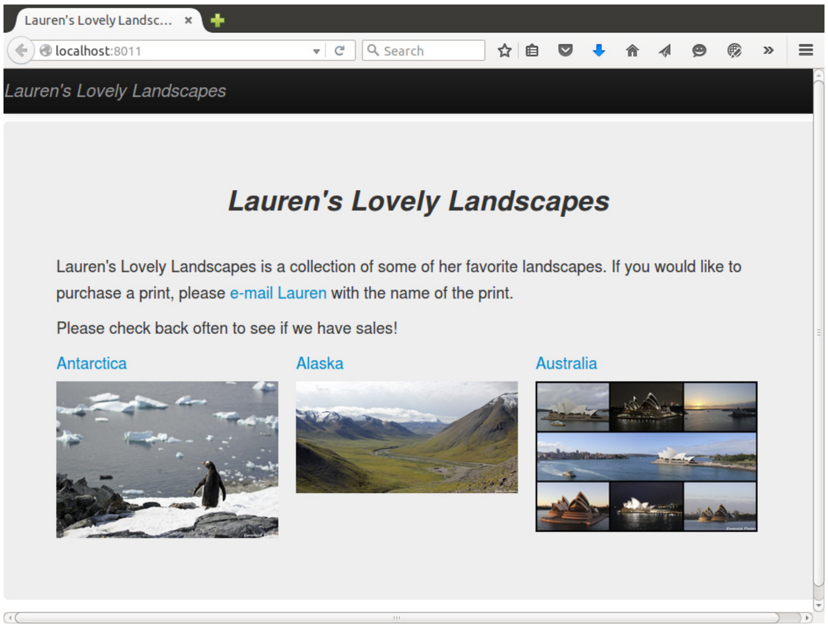
Take a look at the contents of app.js. You'll find the following lines:

```
var appEnv = cfenv.getAppEnv();
app.listen(appEnv.port, appEnv.bind, function() {
   console.log("server starting on " + appEnv.url);
});
```

These lines cause the Express web server to listen on the port specified by the `PORT` environment variable. To set this environment variable in the container, the `docker` command you just ran uses the `--env PORT=8011` option.

In addition, the command's `--net="host"` option causes the container to use the host operating system's network stack internally (purely for development purposes). The app, then, is listening at port 8011 on the host.

Point a browser to http://localhost:8011/, where you can see the Lauren's Lovely Landscapes print store, courtesy of the running container:



Lauren's Lovely Landscapes

Lauren's Lovely Landscapes is a collection of some of her favorite landscapes. If you would like to purchase a print, please e-mail Lauren with the name of the print.

Please check back often to see if we have sales!

You can exit the container at any time to modify code or fix bugs, then run the container again — all within seconds, without cluttering your development environment.

From Node.js to Java EE in seconds

Now that you're convinced that Docker starts and runs a Node.js app in mere seconds, you might wonder how it handles something more heavyweight — Java EE development, say. I'm glad you asked.

Suppose you've just read my "Full-stack Java web dev with Vaadin" tutorial and are eager to try out my Java EE Vaadin example. But you don't have a JDK installed and you don't want to muck up your system by installing one just for an afternoon of tinkering. Docker to the rescue!

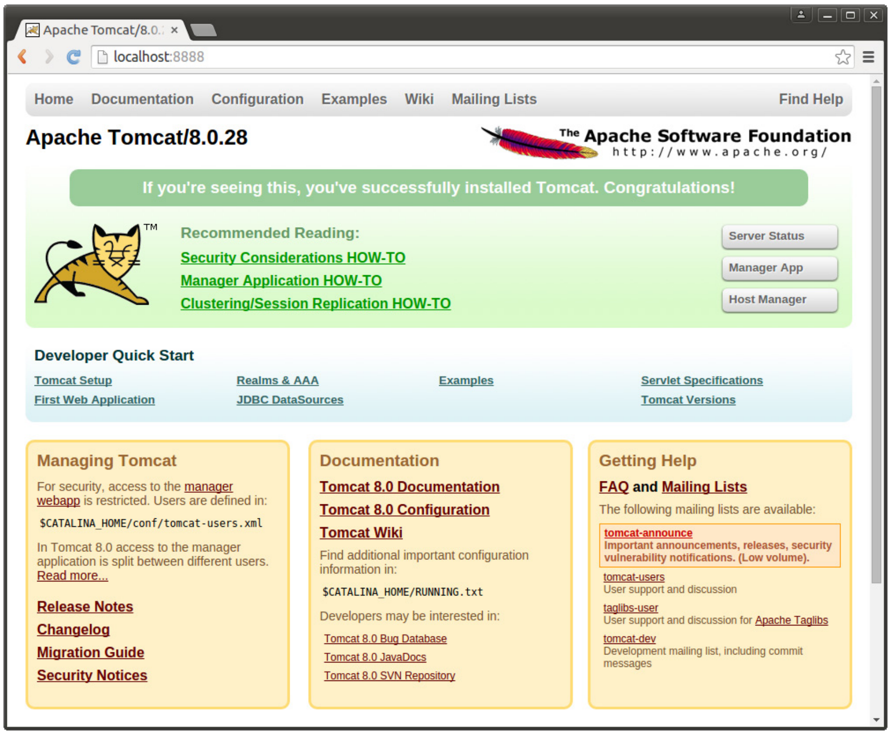
The ArticleViewer.war file in the code download's java directory is an app that comprises several Vaadin UI components from the Vaadin tutorial. You can run ArticleViewer.war on a Tomcat 8 server via Docker. The first step is to start an instance of a Tomcat 8 container from the image you pulled earlier:

```
docker run -it --rm -p 8888:8080 --name=tomcat_server -v $PWD:/mnt tomcat:8
```

This command probably looks familiar to you by now. The container is named `tomcat_server` for easier reference. And your current working directory is mounted as /mnt inside the container.

The `-p 8888:8080` option tells Docker to map the host system's port 8888 to the container's (inside) port 8080. In the `Dockerfile` of the `tomcat` image, you can see that Tomcat is listening on port 8080 inside the container.

Point a browser to http://localhost:8888/, where you can see the familiar Tomcat 8 server greeting page:



Apache Tomcat/8.0.28

If you're seeing this, you've successfully installed Tomcat. Congratulations!

Recommended Reading:
Security Considerations HOW-TO
Manager Application HOW-TO
Clustering/Session Replication HOW-TO

Developer Quick Start
Tomcat Setup
First Web Application
Realms & AAA
JDBC DataSources
Examples
Servlet Specifications
Tomcat Versions

Managing Tomcat
For security, access to the manager webapp is restricted. Users are defined in:
`$CATALINA_HOME/conf/tomcat-users.xml`
In Tomcat 8.0 access to the manager application is split between different users. Read more...
Release Notes
Changelog
Migration Guide
Security Notices

Documentation
Tomcat 8.0 Documentation
Tomcat 8.0 Configuration
Tomcat Wiki
Find additional important configuration information in:
`$CATALINA_HOME/RUNNING.txt`
Developers may be interested in:
Tomcat 8.0 Bug Database
Tomcat 8.0 JavaDocs
Tomcat 8.0 SVN Repository

Getting Help
FAQ and Mailing Lists
The following mailing lists are available:
tomcat-announce
Important announcements, releases, security vulnerability notifications. (Low volume).
tomcat-users
User support and discussion
taglibs-user
User support and discussion for Apache Taglibs
tomcat-dev
Development mailing list, including commit messages

Note that you keep the container running in interactive mode (via the `-it` switch). In that mode, the logs are shown continuously in standard output, which is handy for development.

Attaching to a running Docker container

Now, the mounted directory /mnt inside the container is your current working directory, and ActiveViewer.war is in that directory.

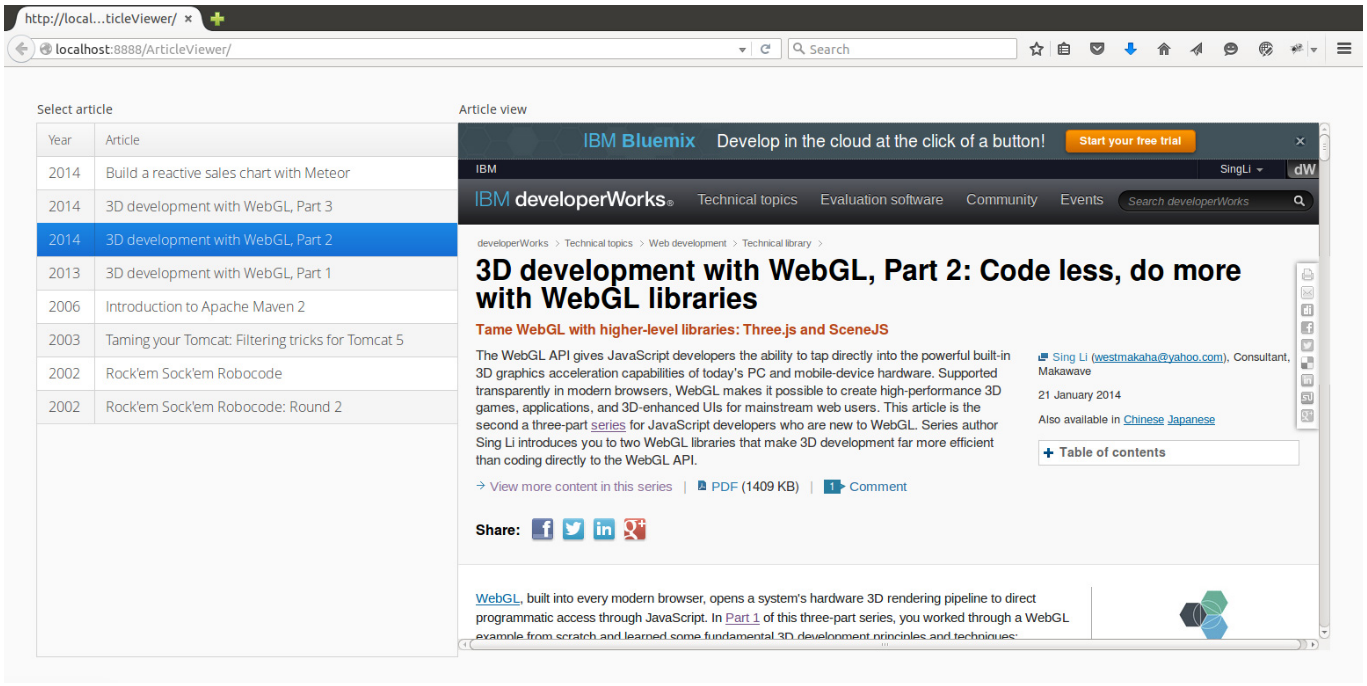
Copy ActiveViewer.war from /mnt to Tomcat 8's webapps directory, where it's automatically deployed:

```
docker exec tomcat_server cp /mnt/ArticleViewer.war /usr/local/tomcat/webapps
```

This time you're not using the `docker run` command, because the `tomcat_server` container is already running. Instead, you attach to the running container and run a command. So you use `docker exec`, supplying the `cp /mnt/ArticleViewer.war /usr/local/tomcat/webapps` command as the final part of the whole command.

Watch the logs in the `tomcat_server` window to view the log trail from the `ArticleViewer` app deployment.

Now point a browser to the app's URL — http://localhost:8888/ArticleViewer/ — to open the running app. Click an article in the list on the left to view the article contents on the right:



Select article

Year	Article
2014	Build a reactive sales chart with Meteor
2014	3D development with WebGL, Part 3
2014	3D development with WebGL, Part 2
2013	3D development with WebGL, Part 1
2006	Introduction to Apache Maven 2
2003	Taming your Tomcat: Filtering tricks for Tomcat 5
2002	Rock'em Sock'em Robocode
2002	Rock'em Sock'em Robocode: Round 2

Article view

IBM Bluemix Develop in the cloud at the click of a button! Start your free trial

IBM SingLi dW

IBM developerWorks Technical topics Evaluation software Community Events Search developerWorks

developerWorks > Technical topics > Web development > Technical library >

3D development with WebGL, Part 2: Code less, do more with WebGL libraries

Tame WebGL with higher-level libraries: Three.js and SceneJS

The WebGL API gives JavaScript developers the ability to tap directly into the powerful built-in 3D graphics acceleration capabilities of today's PC and mobile-device hardware. Supported transparently in modern browsers, WebGL makes it possible to create high-performance 3D games, applications, and 3D-enhanced UIs for mainstream web users. This article is the second a three-part series for JavaScript developers who are new to WebGL. Series author Sing Li introduces you to two WebGL libraries that make 3D development far more efficient than coding directly to the WebGL API.

Sing Li (westmakaha@yahoo.com), Consultant, Makawave

21 January 2014

Also available in Chinese Japanese

+ Table of contents

View more content in this series | PDF (1409 KB) | 1 Comment

Share:

WebGL, built into every modern browser, opens a system's hardware 3D rendering pipeline to direct programmatic access through JavaScript. In Part 1 of this three-part series, you worked through a WebGL example from scratch and learned some fundamental 3D development principles and techniques:

Stop and think for a moment before moving on. You didn't download or install either a JDK or Tomcat 8 on your system. Yet you could run the Vaadin application's code within seconds on a full-fledged Tomcat 8 server — thanks to Docker.

Developing and building Java web apps for Tomcat 8

As you've just seen, you can easily deploy web apps for testing by using Docker. Just as easily, you can compile and build web applications from Java source code via Docker — again, without messing up your development system or taking a long time to install prerequisites.

Change directory to java/LaurenLandscapesJava from the code distribution. In this directory is a Java version of the Lauren's Lovely Landscapes app that you saw in the Node.js example. (Read "Bluemix fundamentals: Deploy a sample Java application to the cloud" if you're interested in a more detailed description of the code.)

The build.xml file is a standard Apache Ant build file that contains instructions for compiling the code and building the WAR file. By using the webratio Apache Ant image, you can quickly compile and build the WAR after code changes (or any other time):

```
docker run -it --rm -v "$PWD":/mnt webratio/ant bash -c 'cd /mnt; ant'
```

Nothing in this `docker run` command is unfamiliar to you by now. The command runs the Ant build tool in the current working directory mounted as /mnt inside the container. Ant compiles all of the Java source code in the src subdirectory and then builds and bundles the WAR file, placing it as lauren.war in the dist subdirectory.

Copy lauren.war from the java/LaurenLandscapesJava/dist directory to the java directory:

```
cp dist/lauren.war  ..
```

You can deploy the lauren.war app to Tomcat 8 with the command:

```
docker exec tomcat_server cp /mnt/lauren.war /usr/local/tomcat/webapps
```

This command attaches to the running `tomcat_server` container and copies the lauren.war file into the webapps subdirectory of Tomcat 8 for automatic deployment.

Point a browser now to http://localhost:8888/lauren/, where you can see the Java version of Lauren's Lovely Landscape running on your Dockerized Tomcat 8.

Switching gears from Java to Python

Suppose that — being a polyglot developer — you want to take a break from Java coding to explore the possibilities that the Python Bottle Web Framework offers. To help you explore Python web app development quickly without installation hassles and dependencies hell, Docker once again saves the day.

Change to the python/Laurens.Lovely.Landscapes directory of the code distribution. In that directory is the Python version of the Lauren's Lovely Landscape app — written for the Bottle web framework. The templates (.tpl files) are in the views subdirectory. (For more information on the code, read "Intro to IBM Bluemix DevOps Services, Part 1: Deploy and update a simple app.") You can also find a version of this code at the associated Bluemix DevOps Services repository.

Run this Python app — without installing Python on your system — via the following Docker command (type it as a single line):

```
docker run -it --rm --name lllpython -p 8000:8000 -v "$PWD":/usr/src/myapp -w
/usr/src/myapp python:3.5 python wsgi.py
```

Now point a browser at http://localhost:8000/, where you can see the running Python version of Lauren's Lovely Landscapes app.

When developing in Python, you can simply change the code and rerun the preceding command to test. Running Docker containers is every bit as fast as running natively installed development tools.

Adding and launching a detached database

Most modern web applications involve three tiers: browser as client; a middle-tier application server such as Express, Tomcat 8, or Bottle; and a back-end database. By using Docker, you can quickly add a database without installing it on your development system.

In this final example, you'll add an Apache CouchDB database server to your development environment. You can use the CouchDB API, which is 100 percent compatible with IBM Cloudant NoSQL Database, to test CouchDB or Cloudant code locally.

Change to the database directory in the source distribution. In this directory is the advanced Node.js version of the Lauren's Lovely Landscapes app. This version fetches information about the print store's inventory from CouchDB (or Cloudant). The web store dynamically changes appearance depending on the inventory of prints available. (You can find out more about the code in "Bluemix fundamentals: Add a Cloudant NoSQL database to your Node.js app.")

To launch an instance of Apache CouchDB, use this Docker command (type it as a single line):

```
docker run -d --name couchdb -p 5984:5984 -e COUCHDB_USERNAME=user -e
COUCHDB_PASSWORD=abc123 -e COUCHDB_DBNAME=prints -v $PWD/data:/data frodenas/couchdb
```

The command uses `COUCHDB_USERNAME`, `COUCHDB_PASSWORD`, and `COUCHDB_DBNAME` environment variables to configure the instance to match the values used in the code. The current working directory is mounted as /data inside the container. CouchDB will write data into this directory, making it possible to restart the container without losing data.

Note that in this example you run the container with the `-d` option instead of `-it --rm` options. The `-d` option starts the CouchDB instance detached. You can use the `docker ps` command to see all the running detached docker containers.

Next, install the app's Node.js dependencies:

```
docker run -it --rm --name llldepends -v "$PWD":/usr/src/myapp -w /usr/src/myapp node:0.10.40 npm install
```

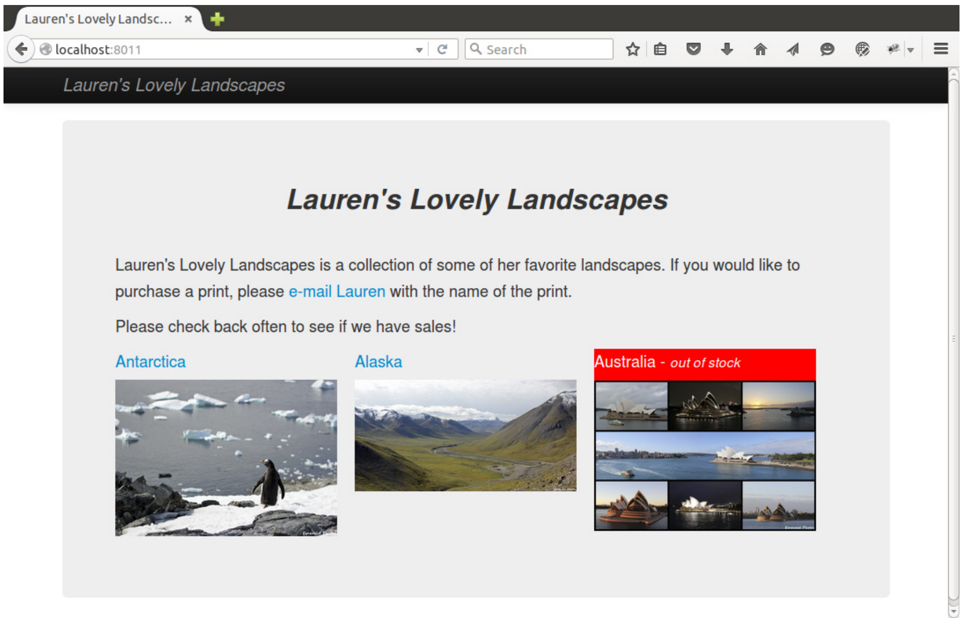
The Apache CouchDB instance doesn't hold any data yet. Use the following command (type it as a single line) to run the dataseeder.js code, which creates a document in the Apache CouchDB instance to populate the store with inventory:

```
docker run -it --rm --name dataseeder --net="host" -v "$PWD":/usr/src/myapp -w
/usr/src/myapp node:0.10.40 node dataseeder.js
```

Run the `lllnode` app at port 8011:

```
docker run -it --rm --name lllnode --env PORT=8011 --net="host" -v "$PWD":/usr/src/myapp
-w /usr/src/myapp node:0.10.40 node app.js
```

Point a browser at http://localhost:8011/ to open the Lauren's Lovely Landscapes store:



Lauren's Lovely Landscapes

Lauren's Lovely Landscapes is a collection of some of her favorite landscapes. If you would like to purchase a print, please e-mail Lauren with the name of the print.

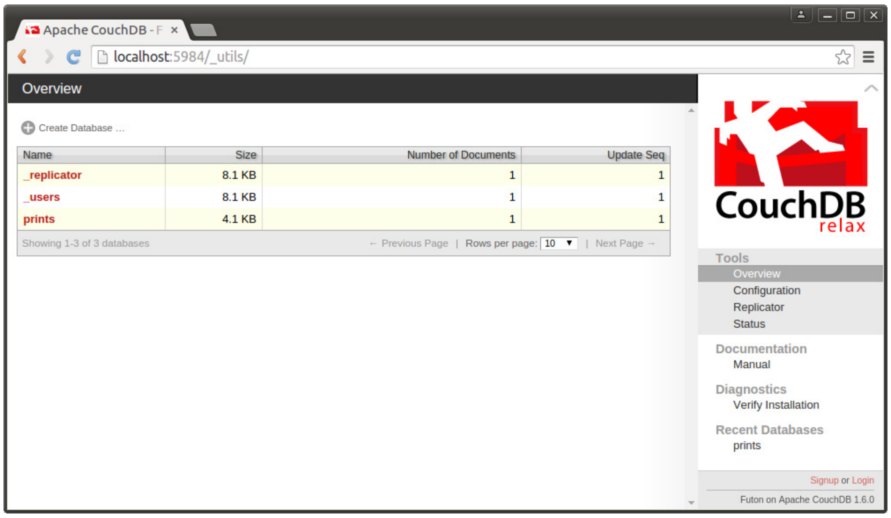
Please check back often to see if we have sales!

Antarctica Alaska Australia - *out of stock*

You can see that the print for Australia is out of stock. That product category is highlighted in red and can't be selected.

To simulate restocking of prints for Australia, you'll access the Apache CouchDB document and modify the inventory level directly.

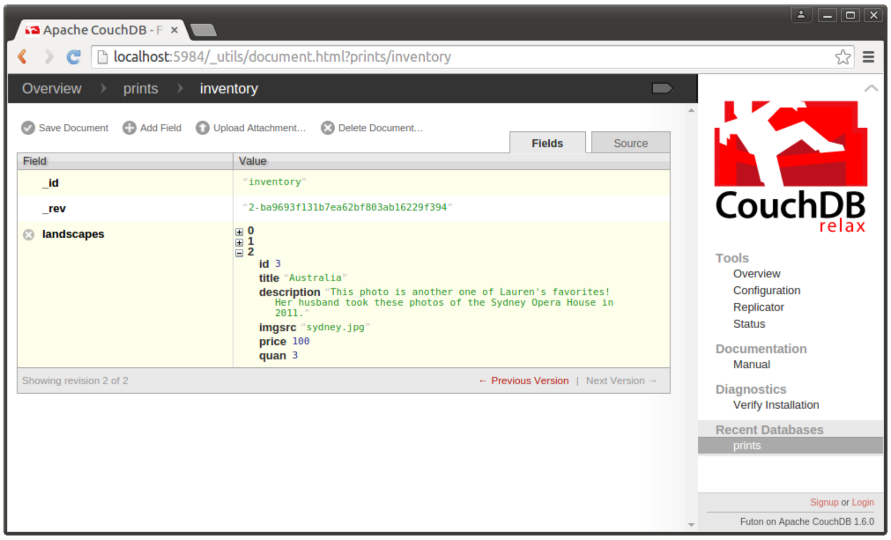
Apache CouchDB offers a simple GUI, named Futon, for accessing stored documents. Point a browser to http://127.0.0.1:5984/_utils/ to open the Futon GUI:



Name	Size	Number of Documents	Update Seq
_replicator	8.1 KB	1	1
_users	8.1 KB	1	1
prints	4.1 KB	1	1

Showing 1-3 of 3 databases ← Previous Page | Rows per page: 10 | Next Page →

In Futon, select `prints`. Open the document and expand inventory item number 3. Double-click the Australia print to open the document for editing. Change the `quan` field from `0` to `3`. Click the small green check-mark icon on the right side, and then click **Save Document** at the top left:



Field	Value
_id	"inventory"
_rev	"2-ba9693f131b7ea62bf803ab16229f394"
landscapes	0 1 2 id 3 title "Australia" description "This photo is another one of Lauren's favorites! Her husband took these photos of the Sydney Opera House in 2011." imgsrc "sydney.jpg" price 100 quan 3

Showing revision 2 of 2 ← Previous Version | Next Version →

With the quantity in the database updated, reload the browser page at http://localhost:8011/. You can see that the store is fully stocked now. The Australia print is no longer marked in red and can be selected.

When you work with three-tiered web applications, Docker streamlines the development workflow for midtier application coding and facilitates instant local deployment of back-end databases for testing.

Conclusion

In this competitive age, you can't afford to live without incorporating Docker in your development workflow. By eliminating the need to repeatedly install and uninstall complex interdependent developer packages and tools, Docker can shorten your development and exploration time.

Take advantage of Docker to boost your polyglot development IQ and dramatically increase your development productivity.

Downloadable resources

Description	Name	Size
Code sample	code_docker.zip	22MB

Related topics

- Docker: Learn more about Docker at the project website.
- Docker Hub: Pull images from or contribute your own images to the community Docker image repository.
- Bottle: Explore Bottle, a lightweight web framework for Python.
- IBM Containers for Bluemix: Find out how to run Docker containers in a hosted cloud environment on IBM Bluemix.
- "Docker for Linux on Power Systems": Get the steps for installing Docker binaries on Linux on Power systems.