Killing Container

- Let's kill all our containers:

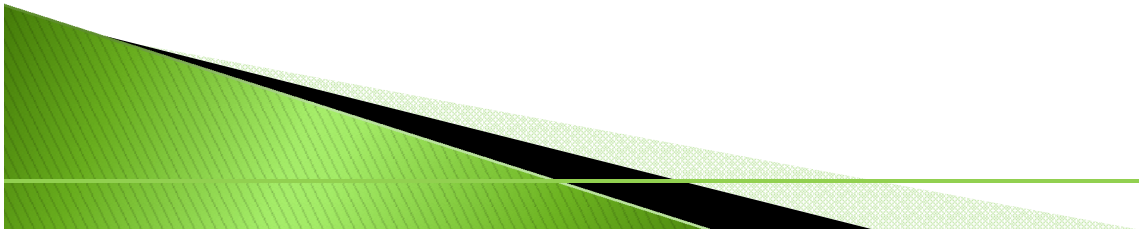
```
docker kill 068 57ad
```

- The stop and kill commands can take multiple container IDs.
- Those containers will be terminated immediately (without the 10 seconds delay).
- Do check running containers now.

List stopped Container

- Let's stopped containers:

```
docker ps -a
```

Atgen

Removing Container

- Let's remove our container:

```
docker rm <yourContainerID>
```

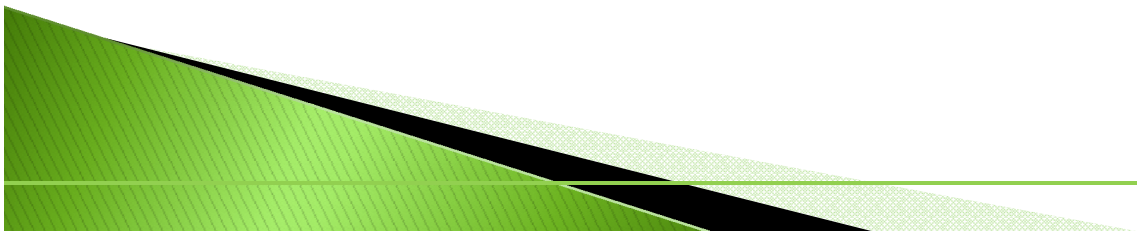
Debugging a Container

- You can SSH into a container using “docker exec”:

```
docker exec -it <yourContainerId> bash
```

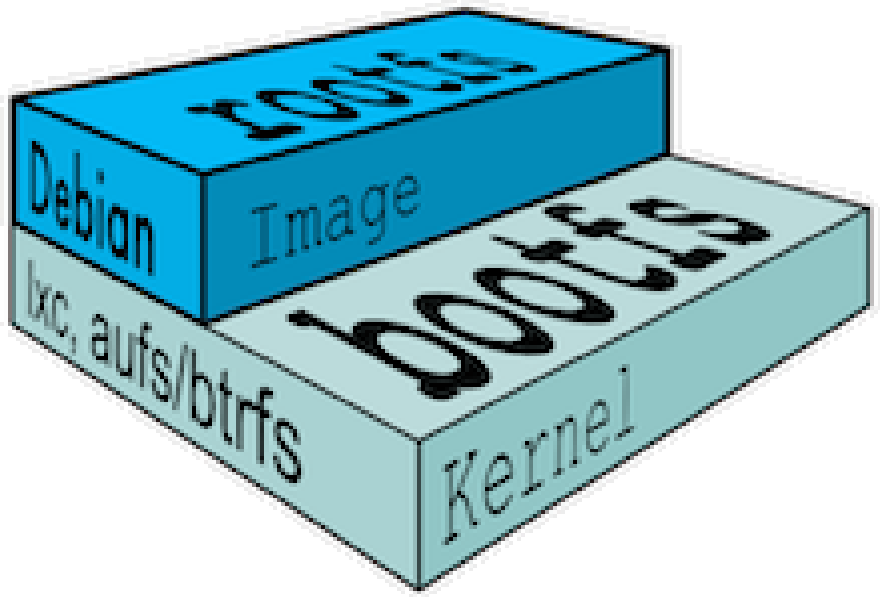
Session: 5

Docker – Images



Atgen

Docker Images



Atgen

What is an image?

- An image is a collection of files + some meta data.
- Images are made of *layers, conceptually stacked on top of each other.*
- Each layer can add, change, and remove files.
- Images can share layers to optimize disk usage, transfer times, and memory use.
- Example:
  - CentOS
  - JRE
  - Tomcat
  - Dependencies
  - Application JAR
  - Configuration

Container vs Image

- An image is a read-only filesystem.
- A container is an encapsulated set of processes running in a read-write copy of that filesystem.
- docker run starts a container from a given image.
- Its “A chicken-and-egg problem”.
- The only way to create an image is by "freezing" a container.
- The only way to create a container is by instantiating an image.

Container vs Image

- An image is a read-only filesystem.
- A container is an encapsulated set of processes running in a read-write copy of that filesystem.
- docker run starts a container from a given image.

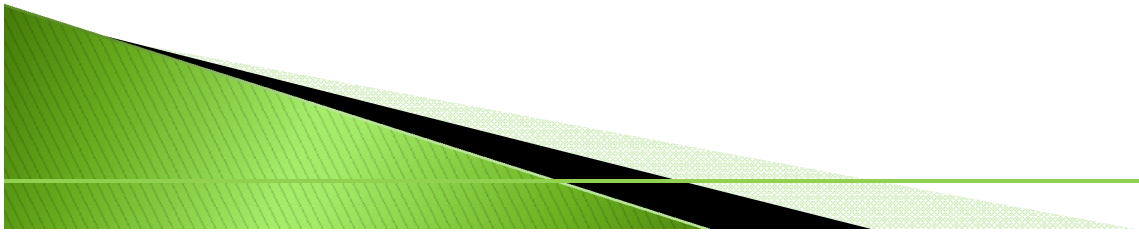
Store & manage images

- Images can be stored:
  - On your Docker host.
  - In a Docker registry.
- You can use the Docker client to download (pull) or upload (push) images.
- To be more accurate: you can use the Docker client to tell a Docker server to push and pull images to and from a registry.

Showing current images

- Let's look at what images are on our host now.

```
docker images
```

Atgen

Searching for images

- We cannot list all images on a remote registry, but we can search for a specific keyword:

```
docker search zookeeper
```

- "Stars" indicate the popularity of the image.

Downloading images

- There are two ways to download images.
  - Explicitly, with “docker pull”.
  - Implicitly, when executing “docker run” and the image is not found locally.

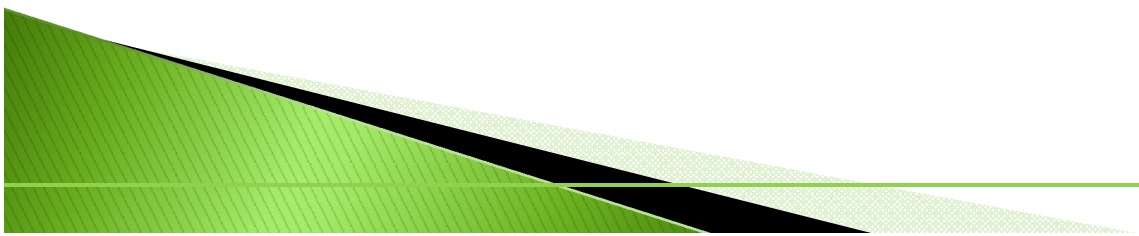
- Pulling an image.

  docker pull debian:jessie

- Images can have tags.
- Tags define image versions or variants.
- “docker pull ubuntu” will refer to “ubuntu:latest”.
- The :latest tag is generally updated often.

Session: 6

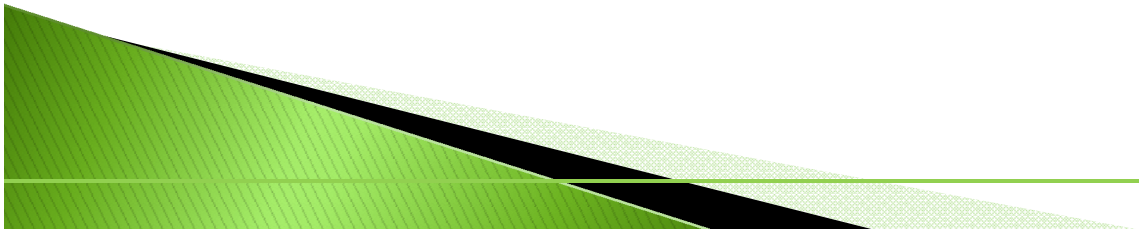
Building Images



Atgen

Building Images Interactively

- Let's have a Use Case:
  - We will build an image that has figlet.
  - First, we will do it manually with docker commit.
  - Then, we will use a Dockerfile and "docker build".



Atgen

Create a new container

- Let's start from base image "ubuntu":

```
docker run -it ubuntu
apt-get update && apt-get install figlet
exit
```

- Inspect the changes:

```
docker diff <yourContainerId>
```

- Commit the changes:

```
docker commit <yourContainerId>
```

Run & Tag the image

- Let's run the new images:

  docker run -it <newImageId>

  figlet hello

- Tagging images:

  docker commit <ContainerId> figlet

- Run it using Tag:

  docker run -it figlet

Dockerfile overview

- A Dockerfile is a build recipe for a Docker image.
- It contains a series of instructions telling Docker how an image is constructed.
- The “docker build” command builds an image from a Dockerfile.

First Dockerfile

- Create a directory to hold our Dockerfile.

```
mkdir myimage
```

- Create a Dockerfile inside this directory.

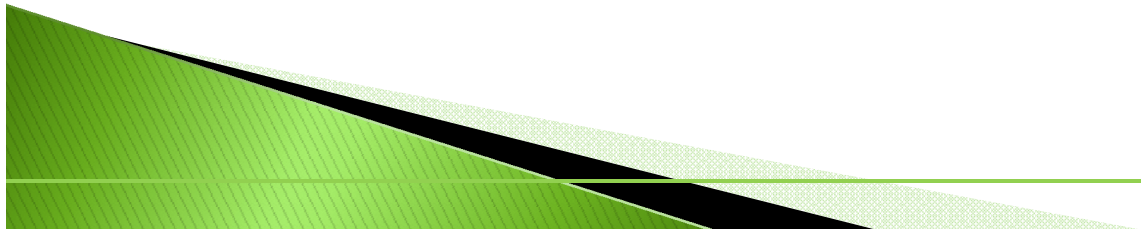
```
cd myimage
vi Dockerfile
```

- Write below in our Dockerfile

```
FROM ubuntu
RUN apt-get update
RUN apt-get install figlet
```

First Dockerfile

- "FROM" indicates the base image for our build.
- Each "RUN" line will be executed by Docker during the build.
- Our "RUN" commands **must be non-interactive.**
- No input can be provided to Docker during the build.



Atgen

First Dockerfile

- Build the Dockerfile:

  docker build -t figlet:1.0 .

- -t indicates the tag to apply to the image.
- . indicates the location of the Directory of Dockerfile.

Run & Tag the image

- Let's run the new images:

  docker run -it <newImageId>

  figlet hello

Using Image & viewing history

- The history command lists all the layers composing an image.
- For each layer, it shows its creation time, size, and creation command.
- When an image was built with a Dockerfile, each layer corresponds to a line of the Dockerfile.

```
docker history figlet
```

Using JSON Syntax - Dockerfile

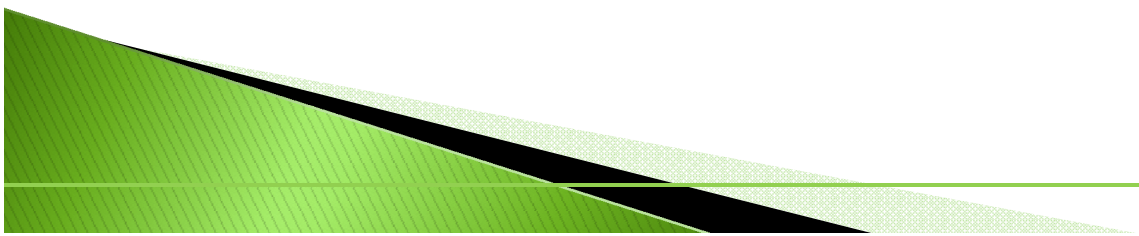
- Change our Dockerfile:

```
FROM ubuntu
RUN apt-get update
RUN ["apt-get", "install", "figlet"]
```

- Now, Build & run it.

Session: 7

Deep Dive – Images



Atgen

Defining a default command

- When people run our container, we want to greet them with a nice hello message, and using a custom font.
- For that, our new Dockerfile will look like:

```
FROM ubuntu
RUN apt-get update
RUN ["apt-get", "install", "figlet"]
CMD figlet -f script hello
```

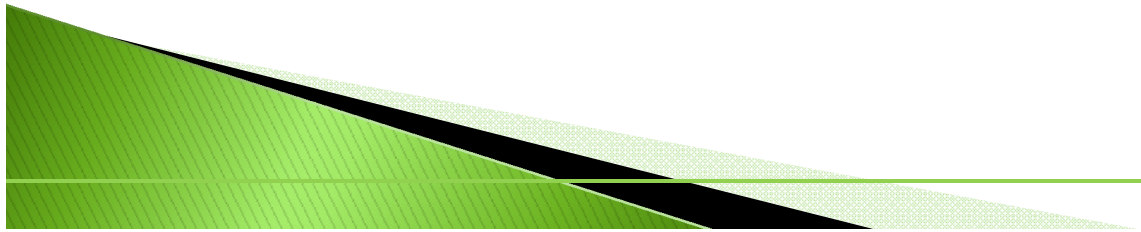
- CMD defines a default command to run when none is given.

Overriding CMD

- If we want to get a shell into our container (instead of running figlet), we just have to specify a different program to run:

```
docker run -it figlet bash
```

- We specified bash.
- It replaced the value of CMD.



Atgen

ENTRYPOINT in Dockerfile

- For that, our new Dockerfile will look like:

```
FROM ubuntu
RUN apt-get update
RUN ["apt-get", "install", "figlet"]
ENTRYPOINT ["figlet", "-f", "script"]
```

- ENTRYPOINT defines a base command (and its parameters) for the container.
- The command line arguments are appended to those parameters.
- Run container of above file like:

```
docker run figlet welcome
```

CMD & ENTRYPOINT

- For that, our new Dockerfile will look like:

```
FROM ubuntu
RUN apt-get update
RUN ["apt-get", "install", "figlet"]
ENTRYPOINT ["figlet", "-f", "script"]
CMD [“welcome to Docker world!”]
```

- Build & Run.

EXPOSE Instruction

- The EXPOSE instruction tells Docker what ports are to be published in this image.

```
EXPOSE 8080
EXPOSE 80 443
EXPOSE 53/tcp 53/udp
```

- All ports are private by default.
- The Dockerfile doesn't control if a port is publicly available.
- A *public port is reachable from other containers and from outside the host.*
- A *private port is not reachable from outside.*

COPY Instruction

- For Use Case, let's build a container that compiles a basic "Hello world" program in C.
- hello.c

```
#include <stdio.h>
int main () {
        puts("Hello, world!");
        return 0;
}
```

- Dockerfile

```
FROM ubuntu
RUN apt-get update
RUN apt-get install -y build-essential
COPY hello.c /
RUN make hello
CMD /hello
```

MAINTAINER Instruction

- The MAINTAINER instruction tells you who wrote the Dockerfile.
- Its optional but recommended.

MAINTAINER Atgen Software <info@atgensoft.com>

RUN – collapsing layers

- It is possible to execute multiple commands in a single step:

```
RUN apt-get update && apt-get install -y wget && apt-get clean
```

COPY Instruction

- The COPY instruction adds files and content from your host into the image.

```
COPY . /src
```

- This will add the contents of the *build context (the directory passed as an argument to* docker build) to the directory /src in the container.

ENV Instruction

- The ENV instruction specifies environment variables that should be set in any container launched from the image.

```
ENV WEBAPP_PORT 8080
```

USER Instruction

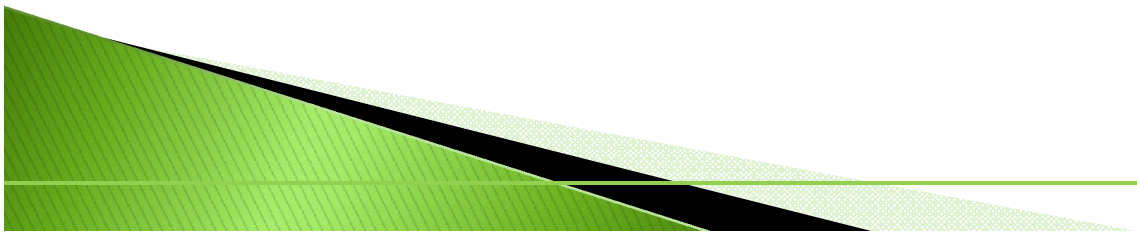
- The USER instruction sets the user name or UID to use when running the image.
- It can be used multiple times to change back to root or to another user.

Uploading images to Docker Hub

- We can share our images through the Docker Hub.
- Below are steps:
  - have an account on the Docker Hub
  - tag our image accordingly (i.e. username/imagename)
  - docker push username/imagename
- Anybody can now docker run username/imagename from any Docker host.

Session: 8

Deep Dive – Containers



Atgen

Default Names – Containers

- When we create a container, if we don't give a specific name, Docker will pick one for us.
- It will be the concatenation of:
  - A mood (furious, goofy, suspicious, boring...)
  - The name of a famous inventor (tesla, darwin, wozniak...)
- Examples: happy_curie, clever_hopper, jovial_lovelace …

Specify Name – Containers

- You can set the name of the container when you create it.

```
docker run --name ticktock jpetazzo/clock
```

- If you specify a name that already exists, Docker will refuse to create the container.
- You can rename containers with “docker rename”.
- This allows you to "free up" a name without destroying the associated container.

Inspecting Containers

- The docker inspect command will output a very detailed JSON map.

  docker inspect <containerID>

- To get specific detail:

  docker inspect --format '{{ json .Created }}' <containerID>

Use Case – Containers

- Run the Docker Hub image nginx, which contains a basic web server:

```
docker run -d -P nginx
```

- Docker will download the image from the Docker Hub.
- -d tells Docker to run the image in the background.
- -P tells Docker to make this service reachable from other computers. (-P is the short version of --publish-all.)

Use Case – Containers

- Finding our WebServer Port:

  docker ps

- Connecting to our web server (GUI):

  http://dockerHostIP:Port

Use Case – Containers

- Containers cannot have public IPv4 addresses.
- They have private addresses.
- Services have to be exposed port by port.
- Ports have to be mapped to avoid conflicts.
- Finding the WebServer Port:

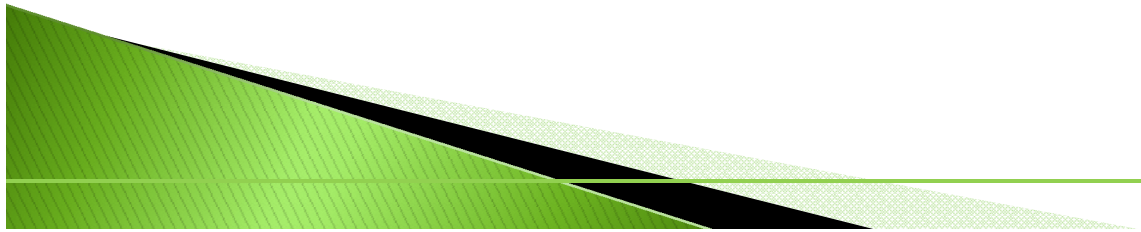
  docker port <containerID> 80

Use Case – Containers

- If you want to set port numbers yourself, no problem:

```
docker run -d -p 80:80 nginx
docker run -d -p 8000:80 nginx
```

- We are running two NGINX web servers.
- The first one is exposed on port 80.
- The second one is exposed on port 8000.
- The convention is port-on-host:port-on-container.

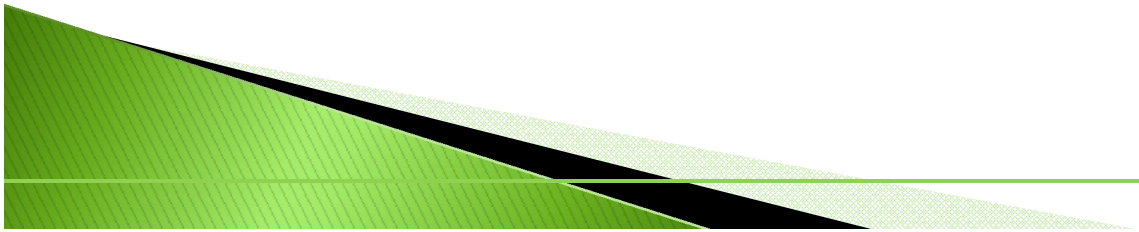


Atgen

Finding IP Address – Containers

- We can use the docker inspect command to find the IP address of the container.

```
docker inspect --format '{{ .NetworkSettings.IPAddress }}' <yourContainerID>
```

Atgen

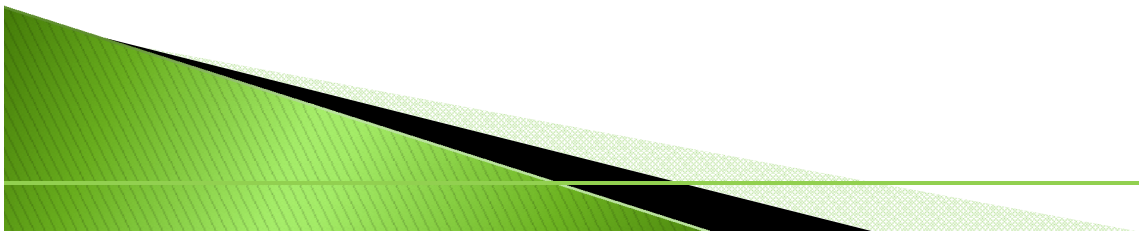
Exercise

- Run a Docker Container with custom “JRE” installed and custom “Tomcat” installed & running in user “tom” home directory “/home/tom” on centos 6.6 platform.

- http://atgensoft.com/html/downloads/Docker_tomcat.zip

Session: 9

Container Network Model



Atgen

Container Network Model

- The CNM was introduced in Engine 1.9.0 (November 2015).
- The CNM adds the notion of a network, and a new top-level command to manipulate and see those networks: “docker network”

```
docker network ls
```

What's in a Network

- Conceptually, a network is a virtual switch.
- It can be local (to a single Engine) or global (across multiple hosts).
- A network has an IP subnet associated to it.
- A network is managed by a *driver.*
- A network can have a custom IPAM (IP allocator).
- Containers with explicit names are discoverable via DNS.

Creating a Network

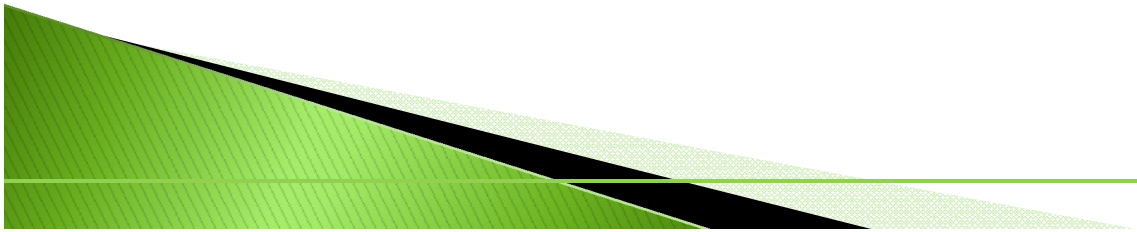
- Let's create a network called dev.

```
docker network create dev
```

- The network can be seen with “network ls” command.

- We will create a named container on this network.

```
docker run -d --name search --net dev tomcat
```

Atgen

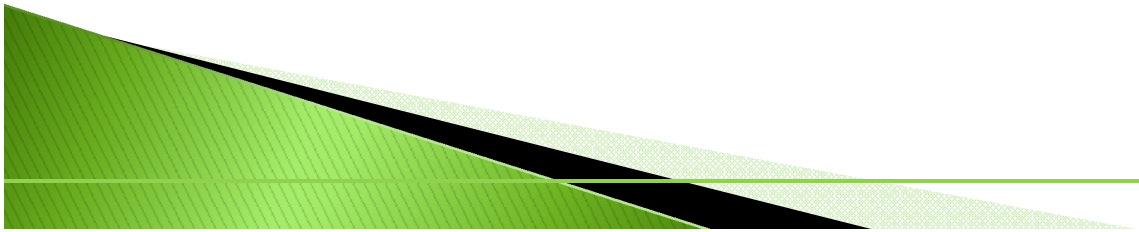
Communication between Containers

- Now, create another container on this network.

```
docker run -ti --net dev alpine sh
```

- From this new container, we can resolve and ping the other one, using its assigned name:

```
ping search
```

Atgen

Connecting multiple Containers

- Another Use Case, to connect containers.
- Let's try to run an application that requires two containers.
- The first container is a web server.
- The other one is a redis data store.
- We will place them both on the dev network created before.

Use Case – Containers

- Running the Web Server:

  docker run --net dev -d -P jpetazzo/trainingwheels

- Check the port that has been allocated to it:

  docker ps -l

- Test the Web Server.

Use Case – Containers

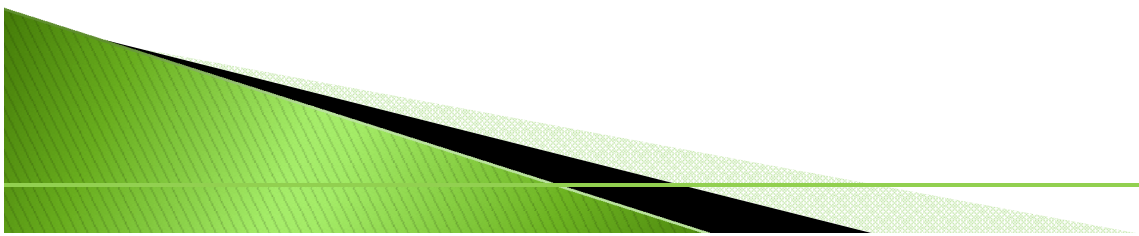
- Start the data store:

  docker run --net dev --name redis -d redis

- Test the Web Server.

Session: 10

Docker – Volumes



Atgen

Working with Volumes

- Docker volumes can be used to achieve many things, including:
  - Bypassing the copy-on-write system to obtain native disk I/O performance.
  - Bypassing copy-on-write to leave some files out of docker commit.
  - Sharing a directory between multiple containers.
  - Sharing a directory between the host and a container.
  - Sharing a single file between the host and a container.

Working with Volumes

- Volumes can be declared in two different ways.
  - Within a Dockerfile, with a VOLUME instruction.

```
VOLUME /uploads
```

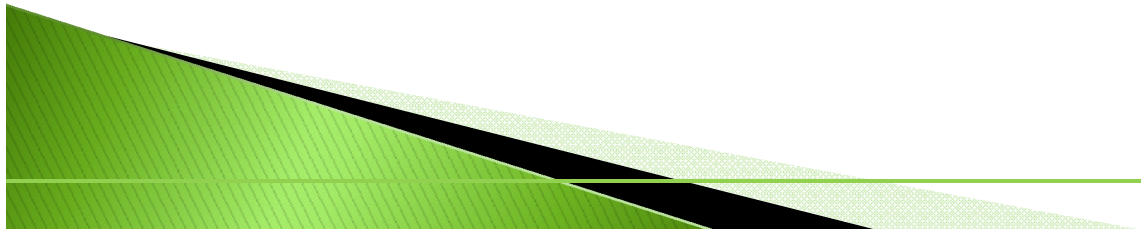
  - On the command-line, with the -v flag for "docker run".

```
docker run -d -v /uploads myapp
```

Sharing Volumes

- You can start a container with exactly the same volumes as another one.
- The new container will have the same volumes, in the same directories.
- They will contain exactly the same thing, and remain in sync.
- Under the hood, they are actually the same directories on the host anyway.
- This is done using the --volumes-from flag for docker run.

```
docker run -it --name alpha -v /var/log:/var/log ubuntu bash
docker run --volumes-from alpha ubuntu cat /var/log/now
```

Atgen

Listing Volumes

- If a container is stopped, its volumes still exist and are available.
- Since Docker 1.9, we can see all existing volumes:

```
docker volume ls
```