#InnoTalks

Starting with Docker, Docker Hub

Docker Cloud

Docker Swarm

Balaji and Jags,
Cloud & DevOps Engineering,
Innominds Software

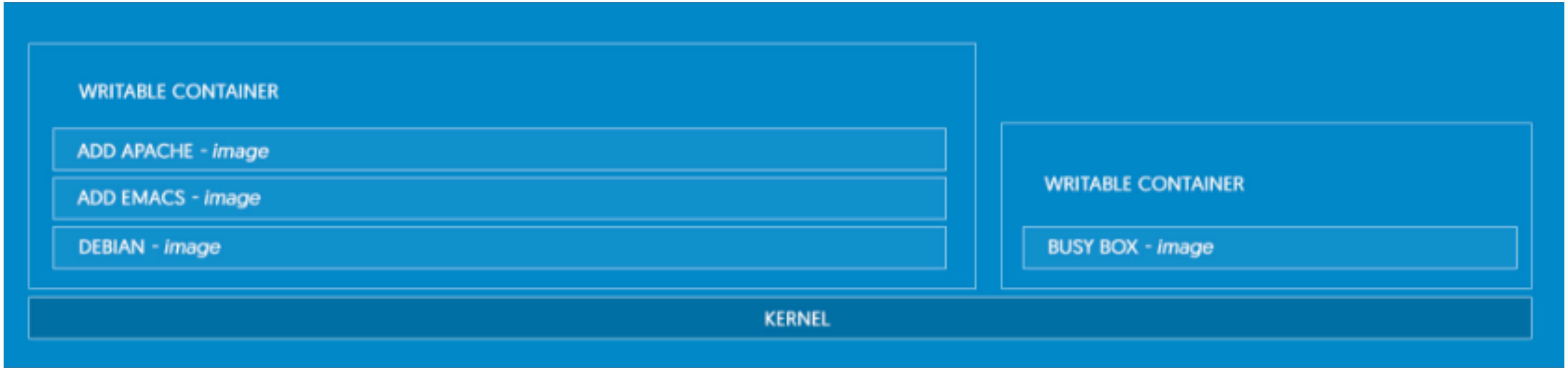
Agenda

- What is Docker?
- Container vs Virtual Machines.
- Docker terminology.
- Introduction to Images.
- Getting started with Containers.
- Dockerfile.
- Docker Hub.
- Docker Cloud.
- Docker Swarm.

What is Docker?

PACKAGE YOUR APPLICATION INTO A STANDARDIZED UNIT FOR SOFTWARE DEVELOPMENT

Docker containers wrap a piece of software in a complete file system that contains everything needed to run: code, runtime, system tools, system libraries – anything that can be installed on a server.
This guarantees that the software will always run the same, regardless of its environment.



LIGHTWEIGHT

Containers running on a single machine share the same operating system kernel; they start instantly and use less RAM. Images are constructed from layered filesystems and share common files, making disk usage and image downloads much more efficient.

OPEN

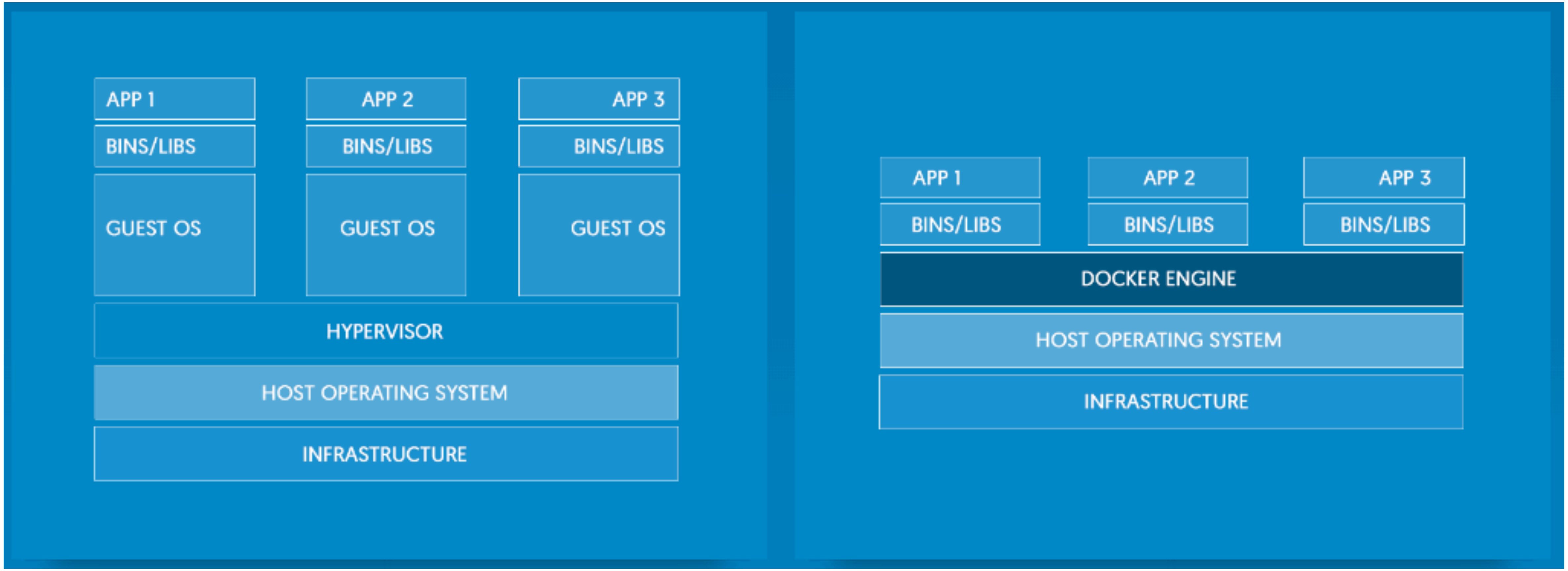
Docker containers are based on open standards, enabling containers to run on all major Linux distributions and on Microsoft Windows -- and on top of any infrastructure.

SECURE BY DEFAULT

Containers isolate applications from one another and the underlying infrastructure, while providing an added layer of protection for the application.

COMPARING CONTAINERS AND VIRTUAL MACHINES

Containers and virtual machines have similar resource isolation and allocation benefits
-- but a different architectural approach allows containers to be more portable and efficient.



Innominds

VIRTUAL MACHINES

Virtual machines include the application, the necessary binaries and libraries, and an entire guest operating system -- all of which can amount to tens of GBs.

CONTAINERS

Containers include the application and all of its dependencies
--but share the kernel with other containers, running as isolated processes in user space on the host operating system.
Docker containers are not tied to any specific infrastructure: they run on any computer, on any infrastructure, and in any cloud.

Docker Terminology

Docker client:
This is what's running in our machine. It's the docker binary that we interface with when we open a terminal and type $ docker pull or $ docker run. It connects to the docker daemon which does all the heavy-lifting, either in the same host (in the case of Linux) or remotely (in our case, interacting with our VirtualBox VM).

Docker daemon:
This is what does the heavy lifting of building, running, and distributing your Docker containers.

Docker Images:
Docker images are the blueprints for our applications. They are our blueprints for actually building a real instance of them. An image can be an OS like Ubuntu, but it can also be an Ubuntu with your web application and all its necessary packages installed.

Docker Container:
Docker containers are created from docker images, and they are the real instances of our containers/lego bricks. They can be started, run, stopped, deleted, and moved.

Docker Hub:
Docker Hub is the official Docker hosted registry that can hold Docker Images. Docker Hub is the official registry.

Innominds

Docker Engine

- Docker is a simple client/server application
- A Docker client talks to a Docker daemon, which execute the work
- Docker executables are written in Go
- Docker daemon also exposes a RESTful API
- Both client and server must be executed as root.

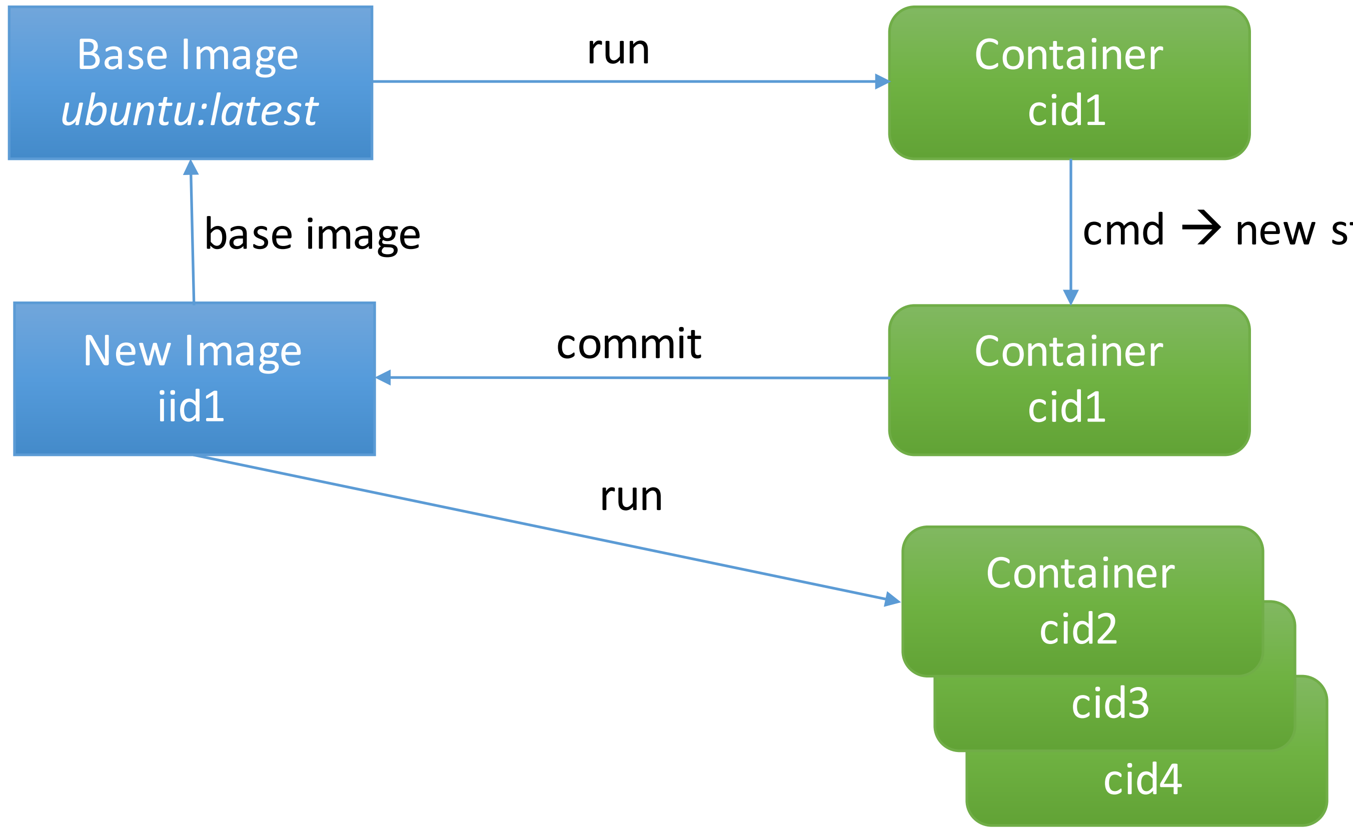
Introduction to Images

- Read-only templates from which containers are launched from.
- Each image consists of a series of layers using the Union File System.
- When an image gets modified, a new layer is created.
- Docker can also use additional file systems.

Getting started with Containers.

- A container is started from an images, which may be locally created, cached locally, or downloaded from registry
- It “looks & feel” like a VM
- Ridiculously fast boot time
- Low resource usage.

Image vs. Container



Docker Commands

- docker search
- docker pull
- docker images
- docker run
- docker ps
- docker Build

Innominds

Docker Run Flags

- **-d** -> to run as a Daemon
- **-t** -> pseudo terminal
- **-i** -> interactive mode
- **--name** -> Container Name
- **-- port** -> to expose ports
- **-v** -> volume mount

START/KILL/REMOVE/ CONTAINERS

- docker stop <container id>
- docker start <container id>
- docker attach <container id>
- docker remove <container id>
- docker rmi <container id>

Containers Communication

- **docker pull redis** -> it will get the Redis image from Docker Hub.
- **docker run -d --name redis1 redis** -> Runs the Redis container with redis1 name
- **docker ps** -> checking the running docker containers, we can see redis running in output.
- **docker run -it --link redis1:redis --name client1 redis sh** -> Here we are connecting containers redis1 and client1

(The Above command takes us to an terminal where we will execute the below commands)

- **ping redis**
- **redis-cli -h redis**

```
redis:6379> PING
PONG
redis:6379> set myvar DOCKER
OK
redis:6379> get myvar
"DOCKER"
redis:6379>
```

Innominds

Container Volumes

- Docker can mount host volumes in read/write mode.
- Data are shared between host and container
- Docker run –it –v /data:/shared ubuntu:14.04 /bin/bash

Dockerfile

- Image representations
- Simple Syntax for describing
- Automate and Script the image creation
- Easy to learn (Look like Shell)
- Fast and reliable

Example Dockerfile

- FROM ubuntu:15.04
- RUN apt-get update
- RUN apt-get install -y apache2
- RUN apt-get install -y apache2-utils
- RUN apt-get install -y vim
- RUN apt-get clean
- EXPOSE 80
- CMD [“apache2ctl”, “-D”, “FORGROUND”]

Docker Build: docker build -t ommudalibalaji/apache:1 .

Docker Compose

Compose is a tool for defining and running multi-container Docker applications. With Compose, you use a Compose file to configure your application's services. Then, using a single command, you create and start all the services from your configuration. To learn more about all the features of Compose see the list of features.

Compose is great for development, testing, and staging environments, as well as CI workflows. You can learn more about each case in Common Use Cases.

Using Compose is basically a three-step process.

- Define your app's environment with a Dockerfile so it can be reproduced anywhere.
- Define the services that make up your app in docker-compose.yml so they can be run together in an isolated environment.
- Lastly, run docker-compose up and Compose will start and run your entire app.

A docker-compose.yml looks like this:

```
version: '2'
services:
  web:
    build: .
    ports:
    - "5000:5000"
    volumes:
    - .:/code
    - logvolume01:/var/log
    links:
    - redis
  redis:
    image: redis
volumes:
  logvolume01: {}
```

Compose has commands for managing the whole lifecycle of your application:

- Start, stop and rebuild services
- View the status of running services
- Stream the log output of running services
- Run a one-off command on a service

Docker Machine

You can use Docker Machine to:

- Install and run Docker on Mac or Windows
- Provision and manage multiple remote Docker hosts
- Provision Swarm clusters

Docker Machine is a tool that lets you install Docker Engine on virtual hosts, and manage the hosts with docker-machine commands. You can use Machine to create Docker hosts on your local Mac or Windows box, on your company network, in your data center, or on cloud providers like AWS or Digital Ocean.

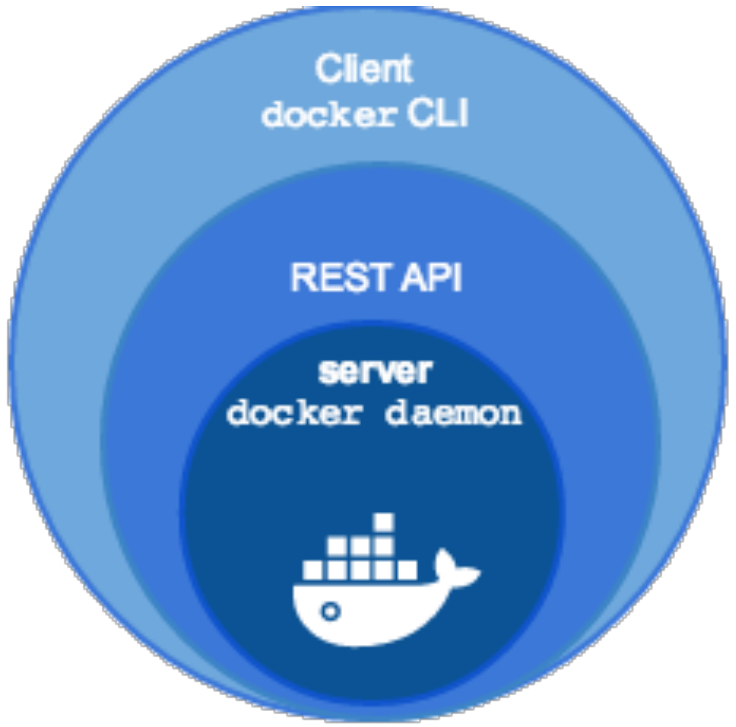
Using docker-machine commands, you can start, inspect, stop, and restart a managed host, upgrade the Docker client and daemon, and configure a Docker client to talk to your host.

Point the Machine CLI at a running, managed host, and you can run docker commands directly on that host. For example, run docker-machine env default to point to a host called default, follow on-screen instructions to complete env setup, and run docker ps, docker run hello-world, and so forth.

What’s the difference between Docker Engine and Docker Machine?

When people say “Docker” they typically mean Docker Engine, the client-server application made up of the Docker daemon, a REST API that specifies interfaces for interacting with the daemon, and a command line interface (CLI) client that talks to the daemon (through the REST API wrapper).

Docker Engine accepts docker commands from the CLI, such as docker run <image>, docker ps to list running containers, docker images to list images, and so on.

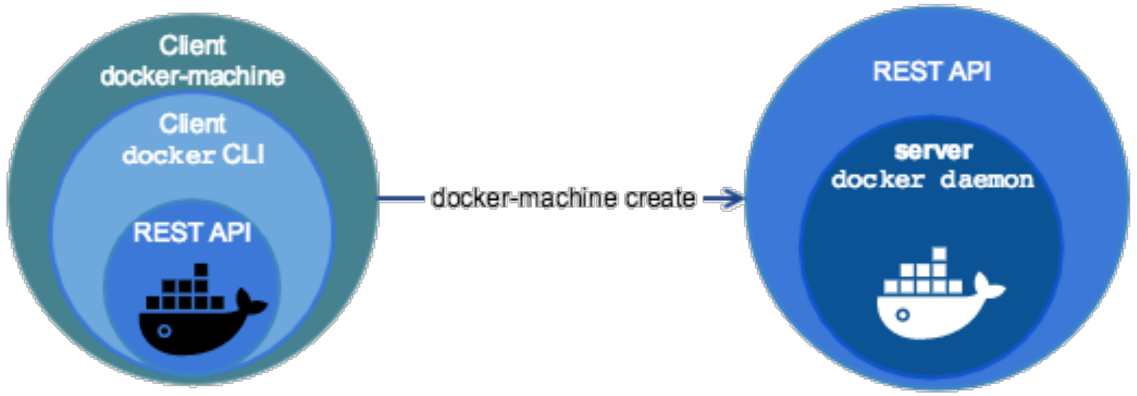


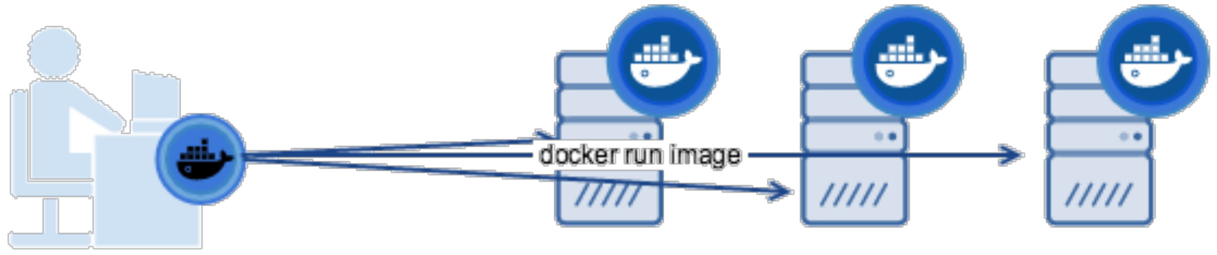
Innominds

Docker Machine is a tool for provisioning and managing your Dockerized hosts (hosts with Docker Engine on them). Typically, you install Docker Machine on your local system. Docker Machine has its own command line clientdocker-machine and the Docker Engine client, docker.

You can use Machine to install Docker Engine on one or more virtual systems. These virtual systems can be local (as when you use Machine to install and run Docker Engine in VirtualBox on Mac or Windows) or remote (as when you use Machine to provision Dockerized hosts on cloud providers).

The Dockerized hosts themselves can be thought of, and are sometimes referred to as, managed "*machines*".





Innominds

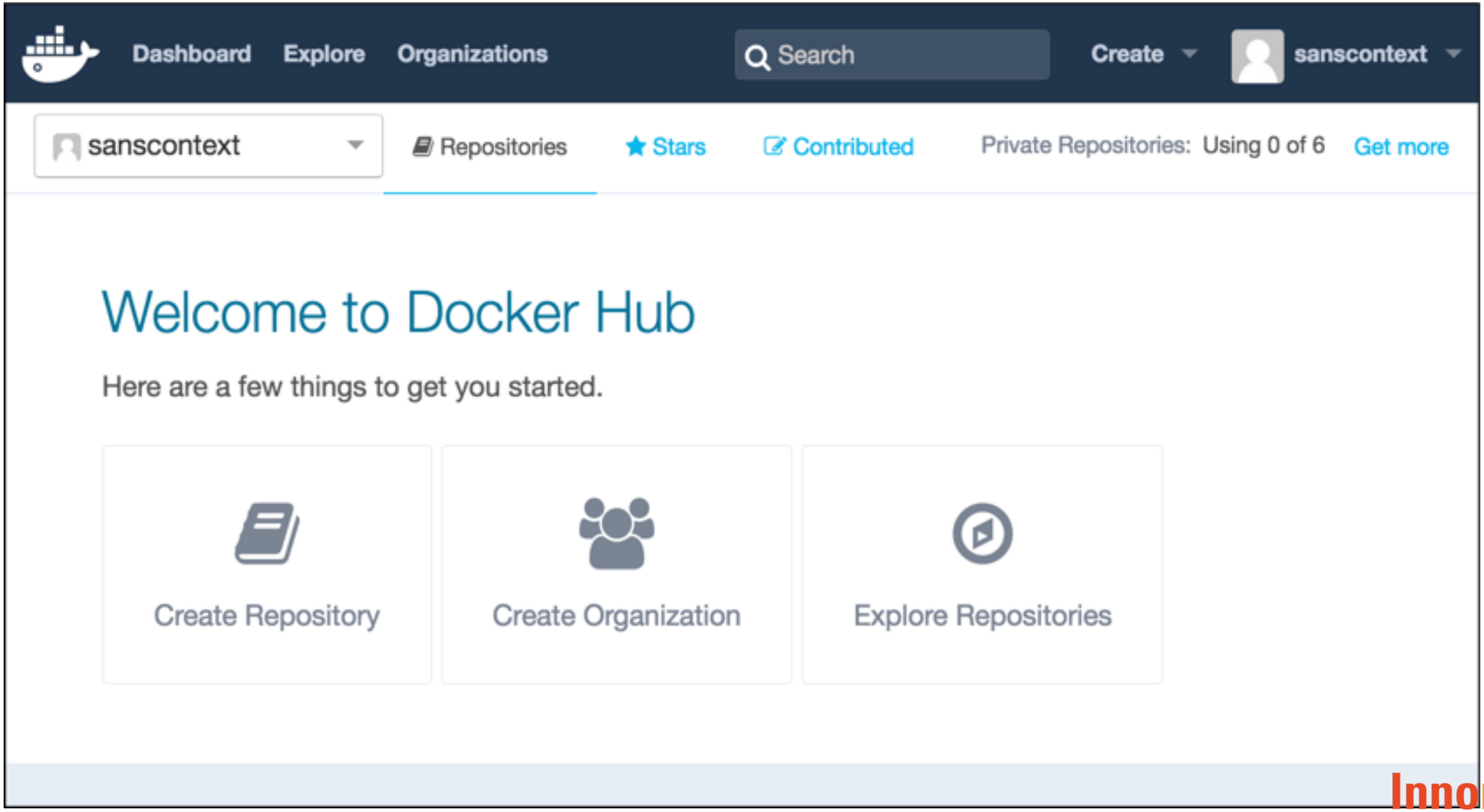
Docker Hub

Overview of Docker Hub

Docker Hub is a cloud-based registry service which allows you to link to code repositories, build your images and test them, stores manually pushed images, and links to Docker Cloud so you can deploy images to your hosts. It provides a centralized resource for container image discovery, distribution and change management, user and team collaboration, and workflow automation throughout the development pipeline.

Innominds

Log in to Docker Hub and Docker Cloud using your free Docker ID.



Welcome to Docker Hub

Here are a few things to get you started.

Innominds

Docker Hub provides the following major features:

Image Repositories: Find, manage, and push and pull images from community, official, and private image libraries.

Automated Builds: Automatically create new images when you make changes to a source code repository.

Webhooks: A feature of Automated Builds, Webhooks let you trigger actions after a successful push to a repository.

Organizations: Create work groups to manage access to image repositories.

GitHub and Bitbucket Integration: Add the Hub and your Docker Images to your current workflows.

Docker commands and Docker Hub
Docker itself provides access to Docker Hub services via the docker search, pull, login, and push commands.

Work with Docker Hub image repositories

Docker Hub provides a place for you and your team to build and ship Docker images.
You can configure Docker Hub repositories in two ways:

Repositories: which allow you to push images from a local Docker daemon to Docker Hub, and

Automated Builds: which link to a source code repository and trigger an image rebuild process on Docker Hub when changes are detected in the source code.

You can create public repositories which can be accessed by any other Hub user, or you can create private repositories with limited access you control.

Docker Cloud

What is Docker Cloud?

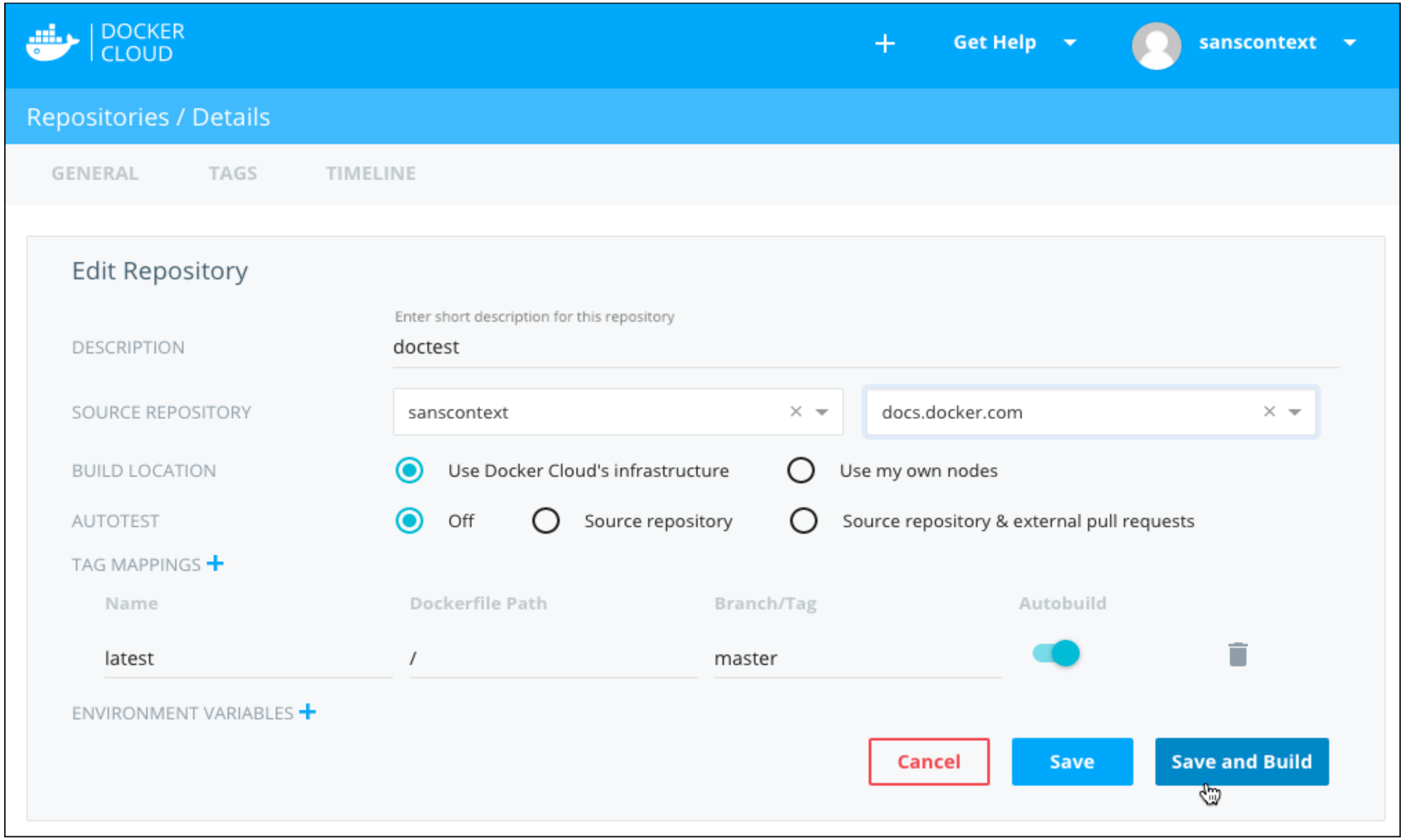
Docker Cloud is a hosted service that provides a Registry with build and testing facilities for Dockerized application images, tools to help you set up and manage your host infrastructure, and deployment features to help you automate deploying your images to your infrastructure.

Your Docker Cloud account and Docker ID

You log in to Docker Cloud using your free Docker ID. Your Docker ID is the same set of credentials you used to log in to Docker Hub, and this allows you to access your Docker Hub repositories from Docker Cloud.

Images, Builds, and Testing

Docker Cloud uses Docker Hub as an online registry service. This allows you to publish Dockerized images on the internet either publicly or privately. Along with the ability to store pre-built images, Docker Cloud can link to your source code repositories and manage building and testing your images before pushing the images.



Repositories / Details

Edit Repository

DESCRIPTION	Enter short description for this repository: doctest
SOURCE REPOSITORY	sanscontext / docs.docker.com
BUILD LOCATION	[✓ Use Docker Cloud's infrastructure] Use my own nodes
AUTOTEST	[✓ Off] Source repository, Source repository & external pull requests

TAG MAPPINGS +

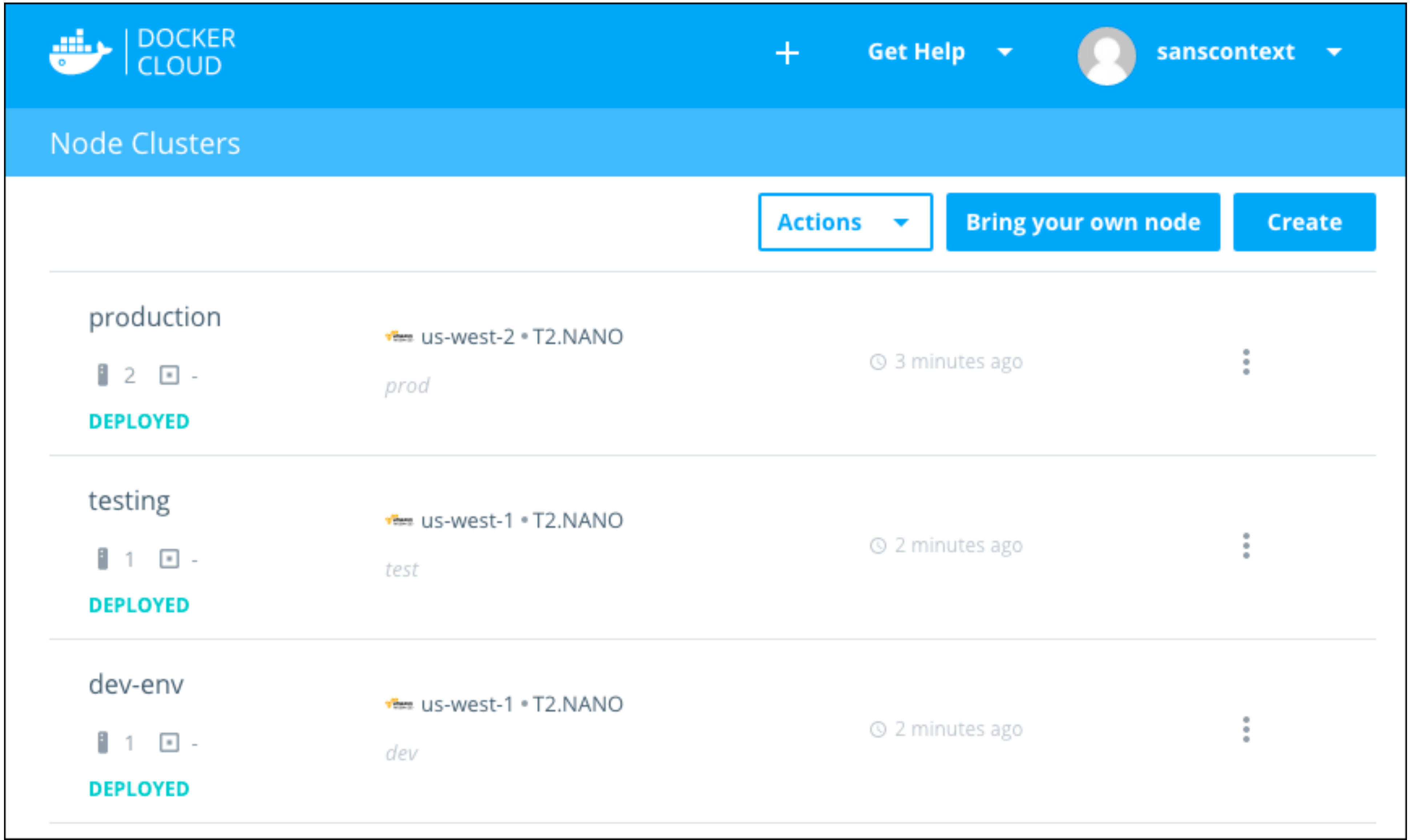
Name	Dockerfile Path	Branch/Tag	Autobuild
latest	/	master	✓

ENVIRONMENT VARIABLES +

Innominds

Infrastructure management

Before you can do anything with images, you need somewhere to run them. Docker Cloud allows you to link to your infrastructure or cloud services provider which lets you provision new nodes automatically, and deploy images directly from your Docker Cloud repositories onto your infrastructure hosts.


DOCKER CLOUD

Get Help

sanscontext

Node Clusters

Actions | Bring your own node | Create

production
us-west-2 • T2.NANO
prod
2 -
DEPLOYED
3 minutes ago

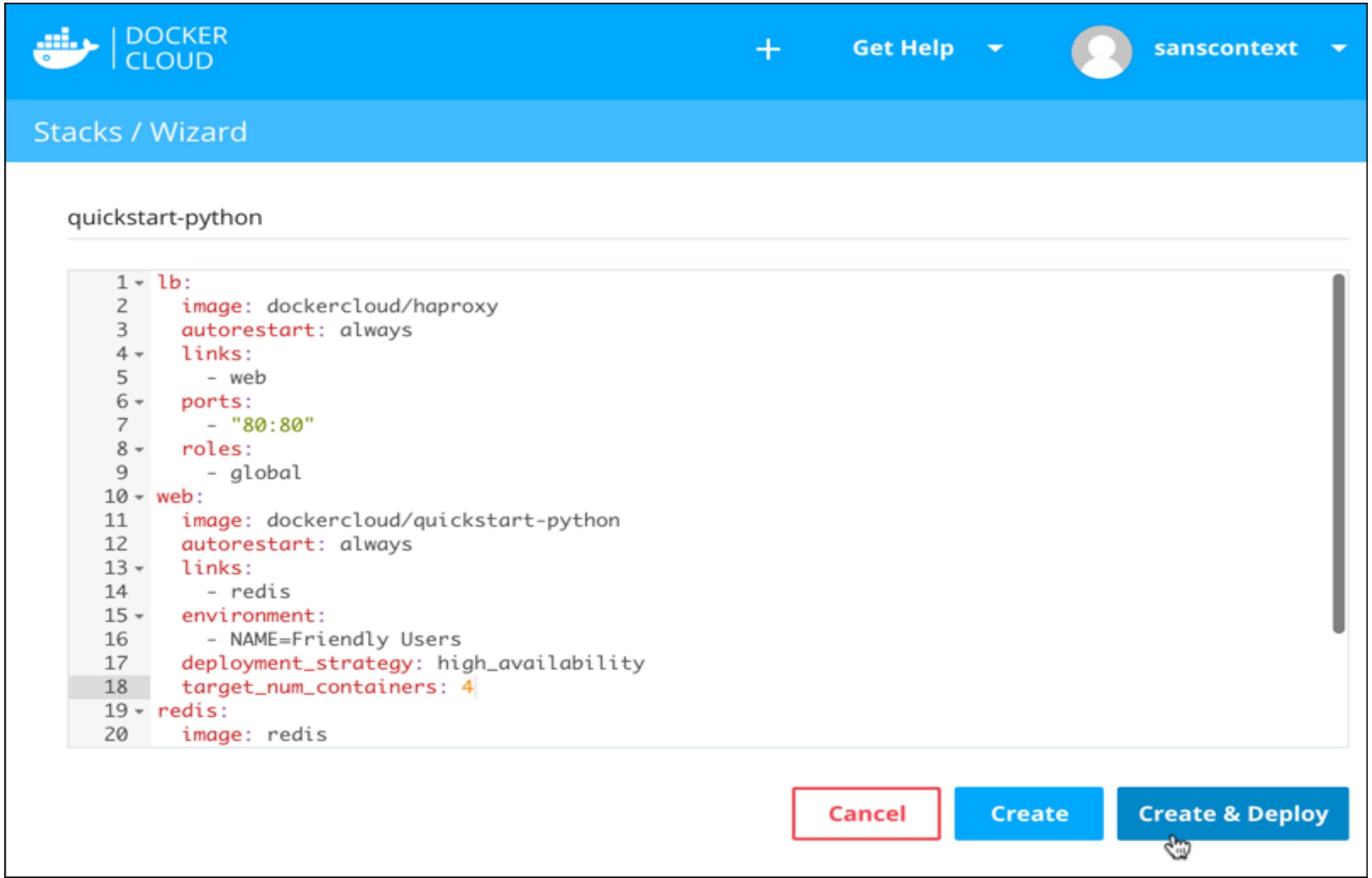
testing
us-west-1 • T2.NANO
test
1 -
DEPLOYED
2 minutes ago

dev-env
us-west-1 • T2.NANO
dev
1 -
DEPLOYED
2 minutes ago

Innominds

Services, Stacks, and Applications

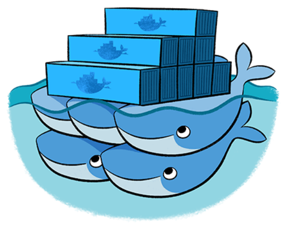
Images are just one layer in containerized applications. Once you've built an image, you can use it to produce containers which make up a service, or use Docker Cloud's stackfiles to combine it with other services and microservices, to form a full application.



quickstart-python

```
lb:
  image: dockercloud/haproxy
  autorestart: always
  links:
    - web
  ports:
    - "80:80"
  roles:
    - global
web:
  image: dockercloud/quickstart-python
  autorestart: always
  links:
    - redis
  environment:
    - NAME=Friendly Users
  deployment_strategy: high_availability
  target_num_containers: 4
redis:
  image: redis
```

Innominds



Docker Swarm

Docker Swarm is native clustering for Docker. It turns a pool of Docker hosts into a single, virtual Docker host. Because Docker Swarm serves the standard Docker API, any tool that already communicates with a Docker daemon can use Swarm to transparently scale to multiple hosts. Supported tools include, but are not limited to, the following:

- Dokku
- Docker Compose
- Docker Machine
- Jenkins

And of course, the Docker client itself is also supported.

Like other Docker projects, Docker Swarm follows the "swap, plug, and play" principle. As initial development settles, an API will develop to enable pluggable backends. This means you can swap out the scheduling backend Docker Swarm uses out-of-the-box with a backend you prefer. Swarm's swappable design provides a smooth out-of-box experience for most use cases, and allows large-scale production deployments to swap for more powerful backends, like Mesos.

Understand Swarm cluster creation

The first step to creating a Swarm cluster on your network is to pull the Docker Swarm image. Then, using Docker, you configure the Swarm manager and all the nodes to run Docker Swarm. This method requires that you:

- open a TCP port on each node for communication with the Swarm manager
- install Docker on each node
- create and manage TLS certificates to secure your cluster

As a starting point, the manual method is best suited for experienced administrators or programmers contributing to Docker Swarm. The alternative is to use docker-machine to install a cluster.

Using Docker Machine, you can quickly install a Docker Swarm on cloud providers or inside your own data center. If you have VirtualBox installed on your local machine, you can quickly build and explore Docker Swarm in your local environment. This method automatically generates a certificate to secure your cluster.

Docker Swarm Commands

Install Swarm

The easiest way to get started with Swarm is to use the official Docker image.

- **docker pull swarm**

Set up Swarm nodes

Each swarm node will run a swarm node agent. The agent registers the referenced Docker daemon, monitors it, and updates the discovery backend with the node's status.

The following example uses the Docker Hub based token discovery service:

Create a Swarm cluster using the docker command.

- **docker run --rm swarm create 6856663cdefdec325839a4b7e1de38e8** # <- this is your unique <cluster_id>

The create command returns a unique cluster id (cluster_id). You'll need this id when starting the Swarm agent on a node.

Docker Swarm Commands

```
root@docker-swarm1:~# docker swarm init
Swarm initialized: current node (0uvgvxftylsuz1b6r8u4yaqpy) is now a manager.

To add a worker to this swarm, run the following command:

docker swarm join \    --token SWMTKN-1-63id3pzxhako4cpf93vpv648qv0tx72xvj4woi3imwudz3nvz9-
eekfchbnqh5gzcgezkklhxuhr \    10.2.0.9:2377

To add a manager to this swarm, run the following command:

docker swarm join \    --token SWMTKN-1-63id3pzxhako4cpf93vpv648qv0tx72xvj4woi3imwudz3nvz9-
5hudo77viwog9fhy39502qgoo \    10.2.0.9:2377

docker service create --name my web --ingress --publish 80:80 nginx
docker network create --driver overlay test-network
docker service create --network ingress --name test-web --publish 80:80 nginx

docker service create --network ingress --name some-rabbit -e RABBITMQ_DEFAULT_USER=user -e
RABBITMQ_DEFAULT_PASS=password --publish 30000:30000 rabbitmq:3-management
```

Innominds

Log into **each node** and do the following.

Start the docker daemon with the -H flag.
This ensures that the docker remote API on *Swarm Agents* is available over TCP for the *Swarm Manager*.

- **docker -H tcp://0.0.0.0:2375 -d**

Register the Swarm agents to the discovery service.
The node's IP must be accessible from the Swarm Manager. Use the following command and replace with the proper node_ip and cluster_id to start an agent:

- **docker run -d swarm join --addr=<node_ip:2375> token://<cluster_id>**

Start the Swarm manager on any machine or your laptop.
The following command illustrates how to do this:

- **docker run -d -p <swarm_port>:2375 swarm manage token://<cluster_id>**

Once the manager is running, check your configuration by running docker info as follows:

- **docker -H tcp://<manager_ip:manager_port> info**