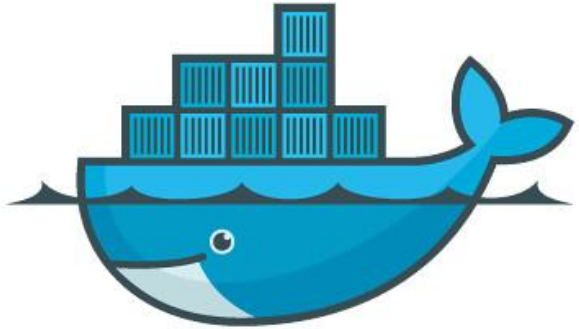


Docker for Network Operators

Anurag Bhatia, Hurricane Electric

Housekeeping rules & announcements!

1. Questions will be taken at regular intervals, usually when one section of the presentation ends. Feel free to type questions in Q&A in meantime. There would also be Q&A and open discussion in the end.

2. We now call these sessions formally as “INNOG Tech sessions” and past tech sessions have been documented here.

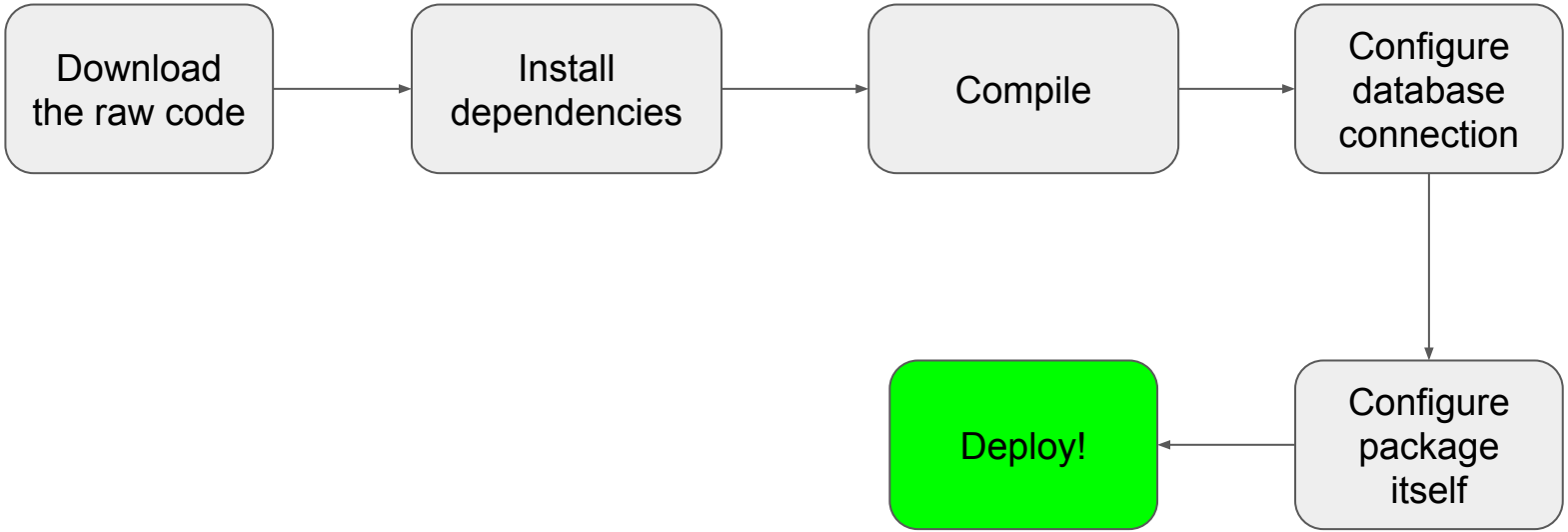
3. These sessions are **not** recorded and that’s by intention. That is done to promote open discussions without worrying about showing up on YouTube videos at some point. Sessions at INNOG annual conference are recorded.

4. Our annual conference INNOG 4 will be from 22nd to 25th June (two day workshop + two day conference). More updates soon on that!

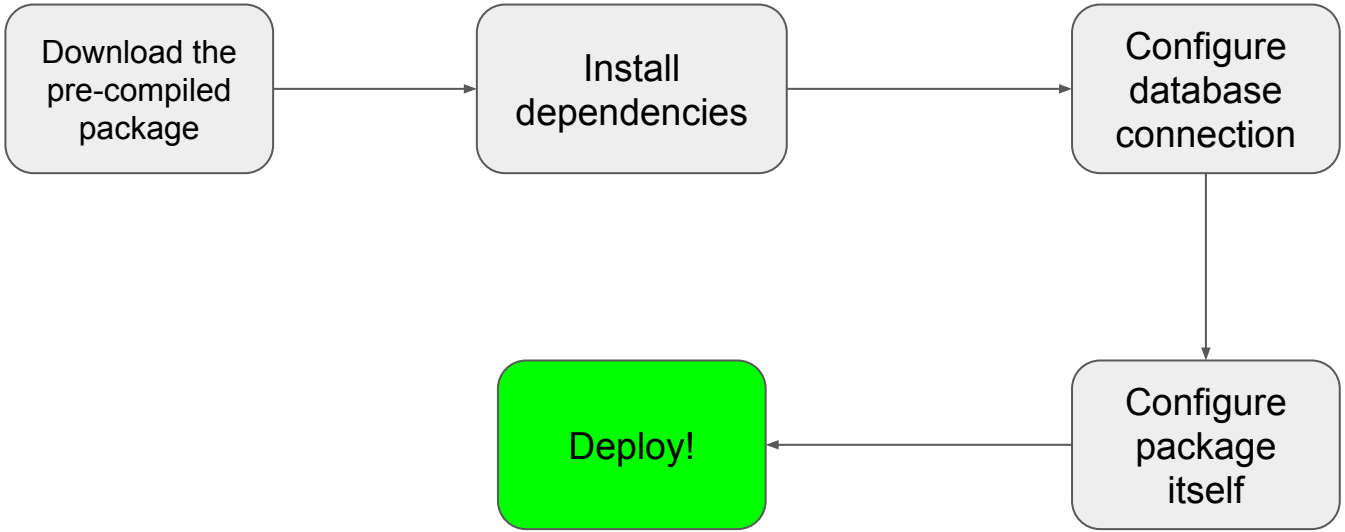
~~What is Docker?~~

How software deployment traditionally works?

Typical traditional software deployment (method 1)



Typical traditional software deployment (method 2)



Challenges with traditional software deployment

1. Takes time to go through documentation, install package and maintain it
2. Migration of software package to a different server requires almost as much effort as to setup fresh
3. Prone to error and mistakes due to large number of steps involved
4. Chance of old problem of “it works on my system” as claimed by developer but not on your system!
5. One application may dependent on one version of some binary, other application may want a different version. Thus dependency conflict can occur.

Screenshare to walk through instructions for setup of Smokeping, Librenms, Zabbix etc

How to set up (in traditional way):

1. Smokeping
2. Librenms
3. Zabbix

Introduction to containers

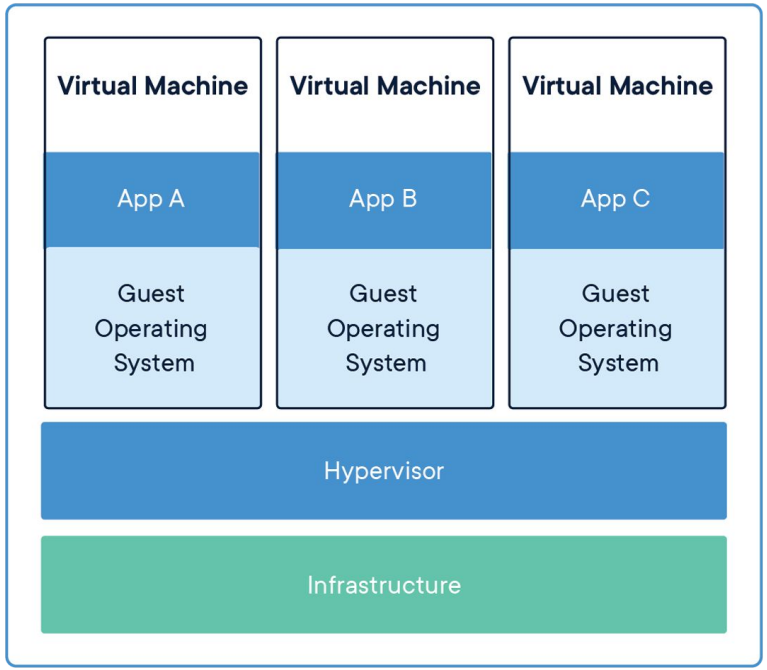
What are Linux Containers?

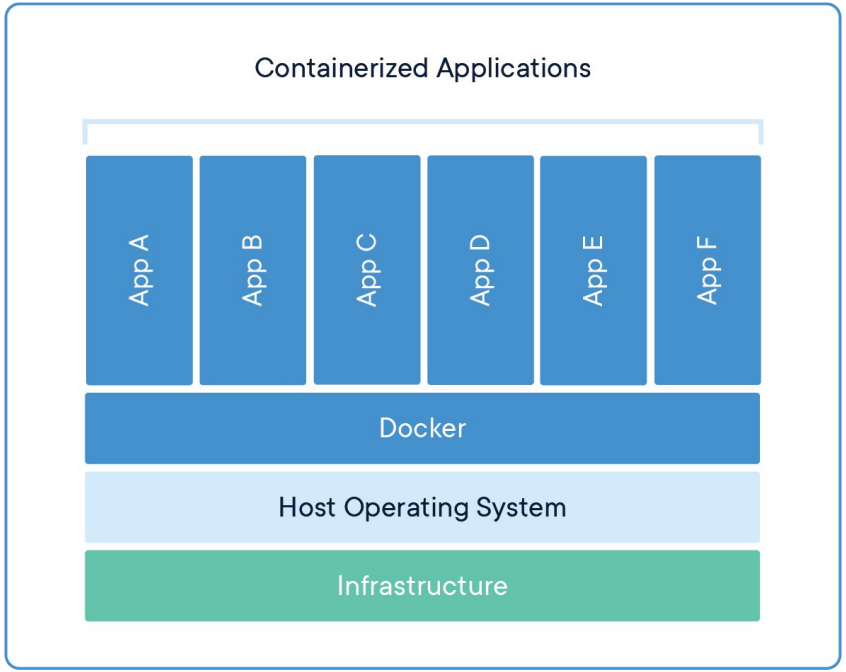
A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another. It also typically has the configuration for dependencies to interact as needed for the working of software.

These are **not** virtual machine though at times might feel very similar. They are extremely lightweight as compared to virtual machines, have low overhead and spin up within seconds instead of minutes. Containers share same Linux kernel & work on the principle of namespaces in Linux.

There are multiple technologies for containerisation and Docker is one of popular ones.

VM Vs Container





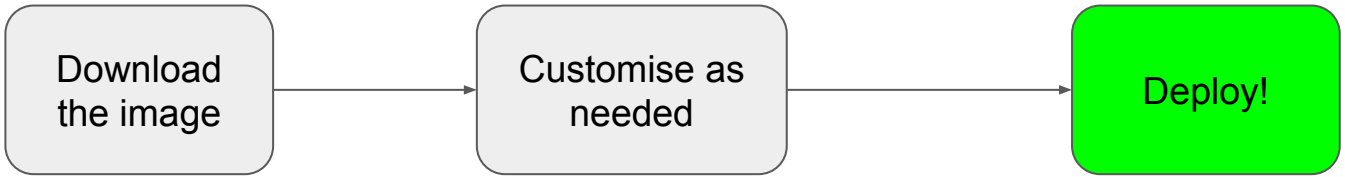
What is docker?

One of few container technologies.

It's one of most popular ones. One needs to have "Docker Engine" installed on the machine to run docker containers. It supports Linux, Windows, Mac etc. Installation instructions here.

Because docker is so popular, many app developers these days ship full fledged working containers with all needed binaries, connections, configurations to use them. That makes deployment extremely fast, easy to manage and update.

Container based software deployment



Docker terms to learn & remember

1. Docker Engine - that's the "engine" which runs on that host on which docker containers run. In order to host a docker container one must have Docker engine installed on the machine.

2. Docker Containers - Essentially the containers which run the applications. Though one can run more than one application in single container but as a good practice it's suggested to run single function per container.

3. Images - These are pre-packages images and can be used to spin up a container. One can also create image based on running container or Dockerfile. A running deployed image is simply a container.

4. Container registry - A registry service hosts container images. Containers images can be pulled in and used.

5. Networks - By default Docker Engine create a bridged network docker0 and that is available for connecting any newly created containers via ad-hoc command. One can create more networks.
   One can bind ports of host to a port on container. E.g port 80 on host can be mapped to port 8080 of container (DNAT)

6. Volumes - One can create a docker volume and attach to a given container

7. Bind mounts - One can mount an existing location on host machine to a location on container. E.g /home/user/data on host can be mounted at /mnt/data inside the containers

Live demonstration of creating docker containers for Ubuntu, CentOS etc.

Basic docker commands

1. docker container list - Lists all running containers
2. docker container list -a - Lists all containers
3. docker network list - Lists docker networks present on the host
4. docker image list - Lists local images present on the docker host
5. docker run --name=A image:latest - Creates container with with name “A” and image with the tag “latest”.
6. docker container stop <container-name>
7. docker container start <container-name>
8. docker container rm <container-name>

Ideas behind running docker containers

- One can get pre-packaged container images & simply spin them up to create containers.
- Containers **do not** store any unique processed data inside instead they store it on a external mounted volume or bind mount. Containers as such are disposable. One can recreate them anytime by simply pulling images again from the registry.
- Typically if an app has dependencies like requirement of database engine etc then those extra services are configured in their own different containers. Thus app goes in it's own container and database engine goes in it's own container. That keeps management extremely easy.
- Port binding is simply destination NAT. If one is exposing say port 80 on host against port 8080 on the container then Docker engine injects rule to destination NAT all traffic incoming on port 80 to port 8080. Docker engine by default SNAT all traffic from containers going to the world.
- Typically containers which are not needed to be exposed to the world are not exposed to the world for security reasons. E.g a database container might be simply connecting to app container. End users access app container and not the database container directly.
- Typically one does not updates a container but instead just downloads new/fresh containers & replace the old one.

Live demo of smokeping via ad-hoc command

(Instructions here)

Idea of environment variables

- Variables defining certain features as supported by the developer
- Can be provided in ad-hoc command or docker-compose or .env file
- Very often included by the developer on instruction page
- Enables features like providing username/password, time zone, etc without having to go inside the container

Smokeping via ad-hoc command

```
docker run -d \
  --name=smokeping \
  -e PUID=1000 \
  -e PGID=1000 \
  -e TZ=Europe/London \
  -p 80:80 \
  -v </path/to/smokeping/config>:/config \
  -v </path/to/smokeping/data>:/data \
  --restart unless-stopped \
    ghcr.io/linuxserver/smokeping
```

Source: https://hub.docker.com/r/linuxserver/smokeping

Challenge with docker ad-hoc commands

- Gets complicated when passing long arguments
- Becomes hard to reproduce a container as one has to remember/keep track of the command used
- If running a complex application needed 3 container then three ad-hoc commands have to be executed
- By default connects all containers to default docker bridge & that does not create network isolation.

Introducing docker-compose

docker-compose.yml

- Single text file in yaml format. Easy to read and include all the instructions needed to setup container(s)
- Includes link to container images
- Includes details about network to attach to. If none-specified, new network is automatically created
- Includes details of volumes to create/attach to or bind mounts to bind to
- Ensures what is supposed to exposed to the world is exposed and what is not supposed to be exposed is not exposed
- Makes it very easy to upgrade containers to their latest images

Rewind...

Ways to deploy container

ad-hoc command (docker run ...)
docker-compose

Live demo of smokeping via docker-compose

Smokeping via docker-compose.yml

```
---
version: "2.1"
services:
  smokeping:
    image: ghcr.io/linuxserver/smokeping
    container_name: smokeping
    environment:
      - PUID=1000
      - PGID=1000
      - TZ=Europe/London
    volumes:
      - </path/to/smokeping/config>:/config
      - </path/to/smokeping/data>:/data
    ports:
      - 80:80
    restart: unless-stopped
```

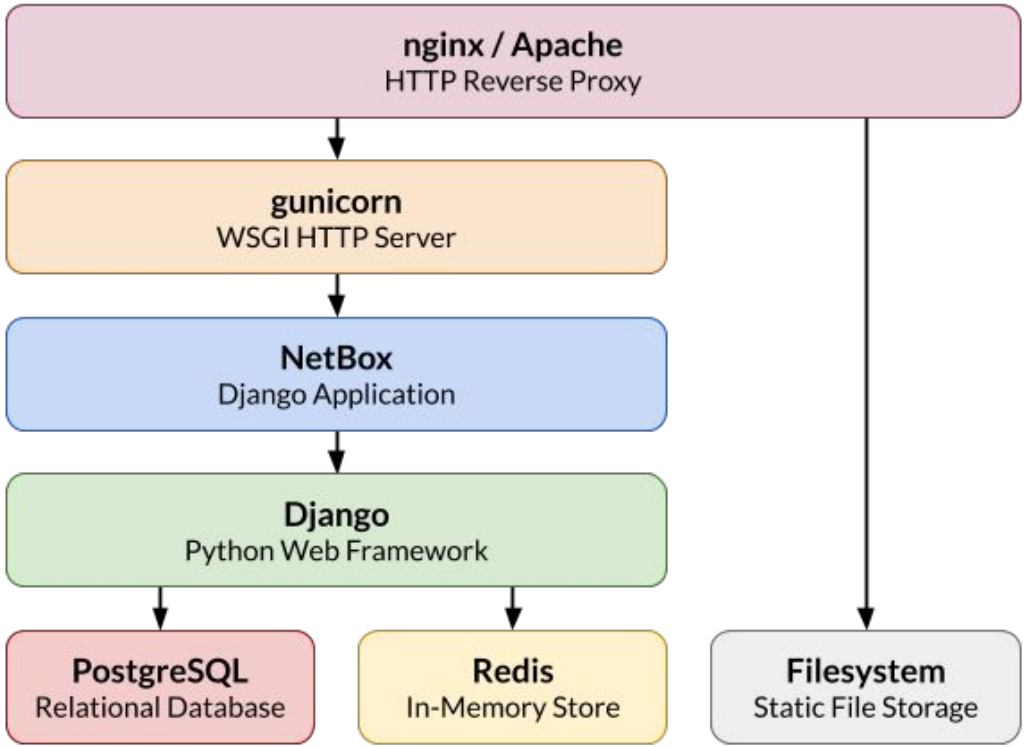
Source: https://hub.docker.com/r/linuxserver/smokeping

Live demo for setup of LibreNMS

(Instructions here)

Netbox stack

https://netbox.readthedocs.io/en/stable/installation/



Live demo for setup of Netbox

(Instructions here)

Live demo for setup of Wireguard

(Instructions here)

How these containers are created?

- Containers are created using some sort of base image OS like Ubuntu, CentOS, Alpine etc. Alpine is smallest image ~ 4MB and has almost nothing installed by default making it very secure.

- Developer creates “Dockerfile” with reference to base image & all the commands which are needed to create that software environment like install x,y,z packages, copy script1, create user etc.

Live demo of a few Dockerfiles & image building

From where should I get container images?

- If you have heard of the developer, know them well then try to get their official container images for security and trust reasons
- If official image is not available, review the Dockerfile of the image published by non-official source
- If it's highly secured environment, build own images based on reviewed dockerfile (though it's OK to use official images as long as one trusts the developer)
- Docker hub for searching for projects: hub.docker.com

What to do about ugly port numbers?

Concept of reverse proxy

1. Sits between outside world and your containers

2. Can map all applications running on any of the ports: 8081, 8092, 9000 etc to a DNS hostname on port 80 and 443

3. Makes it easy to deploy SSL certificates

4. Makes it easy to dual stack public facing services and thus apps become available on IPv6 as well

Live demo of NGINX Proxy Manager for nms.innog.net with SSL

Where should I host my containers?

1. Host locally if you have server capacity available. Certain applications should be hosted locally like NMS, DNS, Speedtest server, smokeping etc.

2. For public facing applications like your website, ticket system, mail server, testing etc you can host locally if compute is easily available or simply host them on a VPS outside of your network. Many options are available for VPS these days.

3. It would be a good idea to have a private network in your server and bind non-public facing docker containers to the private network. You can access that private network directly from your own network Or if server is outside, you can run a layer 3 VPN using Wireguard, IPsec, OpenVPN etc.

Notes on backup

1. Docker containers are reproducible. No need to backup containers unless you have created custom containers.

2. The actual user data is stored using volumes or bind-mounts. In both cases ensure those locations are backed up.

3. One can use tools like Duplicati (comes as a docker container as well) and mount required locations for it to backup.

4. Data storage is extremely cheap these days ~ less than 50paise/GB/month and one can configure Duplicati or similar tool to push regular backups to object storage.

5. Always encrypt data before storing in the cloud.

Demo of bind9

(Instructions here)

Note on IPv6

- Docker does supports IPv6 however overall ecosystem support is not that great.
- When giving publicly routed IPv6 to each container, one should ensure that IPv6 firewall is configured on expected lines.
- For now widely adopted practice is to dual stack frontend proxy container and let it “speak” to backend containers over IPv4. Though this is expected to change over time.

Some cool containers to play with...

- RIPE Atlas - https://hub.docker.com/r/jamesits/ripe-atlas
- HTML 5 speedtest - https://hub.docker.com/r/adolfintel/speedtest
- iperf3 - https://hub.docker.com/r/networkstatic/iperf3
- Nextcloud - https://hub.docker.com/_/nextcloud
- Docker-speedtest-grafana - https://github.com/frdmn/docker-speedtest-grafana
- Kerberos - https://doc.kerberos.io/opensource/installation#docker
- Linux-server.io - Many great images actively maintained by the open source community

Questions?

anurag@he.net
Web: he.net