



# VIRTUALISATION AND CONTAINERS

**20MCA168**

**Amal K Jose**

Assistant Professor, Dept. of Computer Applications  
Amal Jyothi College of Engineering, Kanjirappally

### Syllabus

Working with remote repositories, Security and isolation, Troubleshooting, Monitoring and alerting, Controlling running containers, Containers in a business context

- Working with remote repositories
- Security and isolation
- Troubleshooting
- Monitoring and alerting
- Controlling running containers
- Containers in a business context

## Working with Remote Repositories

- **Docker Hub repositories** allow you to share container images with your team, customers, or the Docker community at large.
- Docker images are pushed to Docker Hub through the **docker push** command.
- **A single Docker Hub repository can hold many Docker images (stored as tags).**
- The power of Docker images is that they're lightweight and portable—they can be moved freely between systems.
- We can easily create a set of standard images, store them in a repository on our network, and share them throughout your organization.
- There have various mechanisms for sharing Docker container images in public and private.
- The most prominent among these is Docker Hub, the company's public exchange for container images.
- Many open-source projects provide official versions of their Docker images there, making it a convenient starting point for creating new containers by building on existing ones, or just obtaining **stock versions** of containers to spin up a project quickly.

### Explore Docker Hub

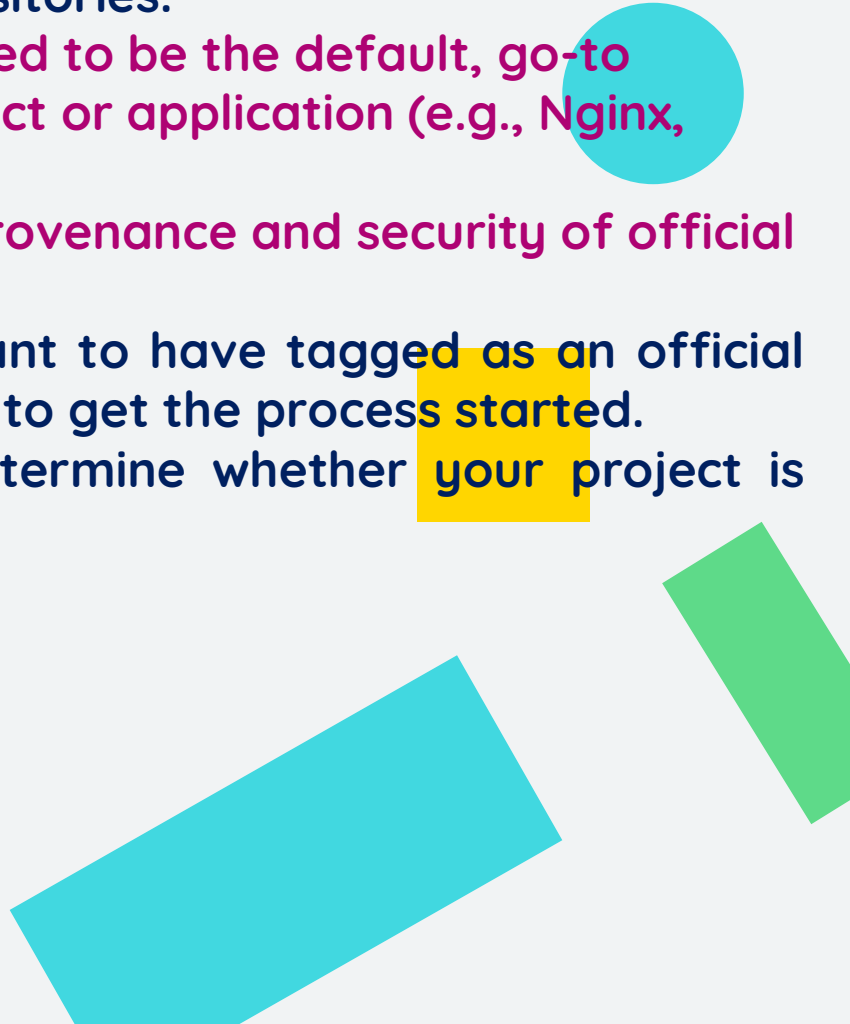
- The easiest way to explore Docker Hub is simply to browse it on the web.
- From the web interface, you can search for publicly available containers by name, tag, or description.
- From there, everything you need to download, run, and otherwise work with container images from Docker Hub comes included in the open-source version of Docker—chiefly, the `docker pull` and `docker push` commands.

### Docker Hub organizations for teams

- If we are using Docker Hub with others, we can create an organization, which allows a group of people to share specific image repositories.
- **Organizations** can be further subdivided into **teams**, each with their own sets of repository privileges.
- Owners of an organization can create new teams and repositories, and assign repository read, write, and admin privileges to fellow users.

### Docker Hub repositories

- Docker Hub repositories can be public or private. Public repos can be searched and accessed by anyone, even those without a Docker Hub account.
  - ❖ Private repos are available only to users you specifically grant access to, and they are not publicly searchable. Note that you can turn a private repo public and vice versa.
- Note also that if you make a private repo public, you'll need to ensure that the exposed code is licensed for use by all.
  - ❖ Docker Hub does not offer any way to perform automatic license analysis on uploaded images; that's all on you.
- While it is often easiest to search a repository using the web interface, the Docker command line or shell also allows you to search for images.
  - ❖ Use `docker search` to run a search, which returns the names and descriptions of matching images.

- Certain repositories are tagged as official repositories.
    - ❖ These provide curated Docker images intended to be the default, go-to versions of a container for a particular project or application (e.g., Nginx, Ubuntu, MySQL).
    - ❖ Docker takes additional steps to verify the provenance and security of official images.
  - If you yourself maintain a project that you want to have tagged as an official repository on Docker Hub, make a pull request to get the process started.
  - Note, however, that it is up to Docker to determine whether your project is worthy of being included.
- 

### Docker push and Docker pull

- Before we can push and pull container images to and from the Docker Hub, we must connect to the Docker Hub with the docker login command, where we can submit our Docker Hub username and password.
  - ❖ By default, docker login takes to Docker Hub, but we can use it to connect to any compatible repository, including privately hosted ones.
- Generally, working with Docker Hub from the command line is fairly straightforward.
  - ❖ Use docker search as described above to find images, docker pull to pull an image by name, and docker push to store an image by name.
  - ❖ A docker pull pulls images from Docker Hub by default unless you specify a path to a different registry.
- Note that when we push an image, it's a good idea to tag it beforehand.
  - Tags are optional, but they help you and your team disambiguate image versions, features, and other characteristics.
  - A common way to do this is to automate tagging as part of your image build process (for instance adding version or branch information.)

### Automated builds on Docker Hub

- Container images (hosted on Docker Hub) can be built automatically from their components hosted in a repository.
  - ❖ Any changes to the code in the repo are automatically reflected in the container; you don't have to manually push a newly built image to Docker Hub.
- Automated builds work by linking an image to a build context, i.e. a repo containing a Docker file that is hosted on a service like GitHub or Bitbucket.
  - Although Docker Hub limits you to one build every five minutes, and there's no support yet for Git large files or Windows containers.
- In a paid Docker Hub account, we can take advantage of parallel builds. (five parallel builds can build containers from up to five different repositories)
  - Each individual repository is allowed only one container build at a time; the parallelism is across repos rather than across images in a repo.

### Webhooks

- While a certain event takes place involving a repository—an image is rebuilt, or a new tag is added— Docker Hub can send a POST request to a given endpoint.
- We can use webhooks to automatically deploy or test an image whenever it is rebuilt, or to deploy the image.



- To use Docker safely, we must be aware of the potential security issues and the major tools and techniques for securing container-based systems.
- We begins by exploring some of the issues surrounding the security of container-based systems that you should be thinking about when using containers.

### Things to Worry About

#### Kernel exploits

- Unlike in a VM, the kernel is shared among all containers and the host, magnifying the importance of any vulnerabilities present in the kernel.
- Should a container cause a kernel panic, it will take down the whole host.
- In VMs, the situation is much better: an attacker would have to route an attack through both the VM kernel and the hypervisor before being able to touch the host kernel.

#### Denial-of-service (DoS) attacks

- All containers share kernel resources. If one container can monopolize access to certain resources—including memory and more esoteric resources such as user IDs (UIDs)—it can starve out other containers on the host, resulting in a DoS where legitimate users are unable to access part or all of the system.

### Container breakouts

- An attacker who gains access to a container should not be able to gain access to other containers or the host. Because users are not namespaced, any process that breaks out of the container will have the same privileges on the host as it did in the container; if you were root in the container, you will be root on the host.
- This also means that you need to worry about potential privilege escalation attacks—where a user gains elevated privileges such as those of the root user, often through a bug in application code that needs to run with extra privileges.

### Poisoned images

- How do you know that the images you are using are safe, haven't been tampered with, and come from where they claim to come from?
- If an attacker can trick us into running her image, both the host and our data are at risk.
- Similarly, we want to be sure the images we are running are up to date and do not contain versions of software with known vulnerabilities.

### Compromising secrets

- When a container accesses a database or service, it will likely require some secret, such as an API key or username and password. An attacker who can get access to this secret will also have access to the service.
- This problem becomes more acute in a microservice architecture in which containers are constantly stopping and starting, as compared to an architecture with small numbers of long-lived VMs.
- The simple fact is that both Docker and the underlying Linux kernel features it relies on are still young and nowhere near as battle hardened as the equivalent VM technology. For the time being at least, containers do not offer the same level of security guarantees as VMs.

### Defense-in-Depth

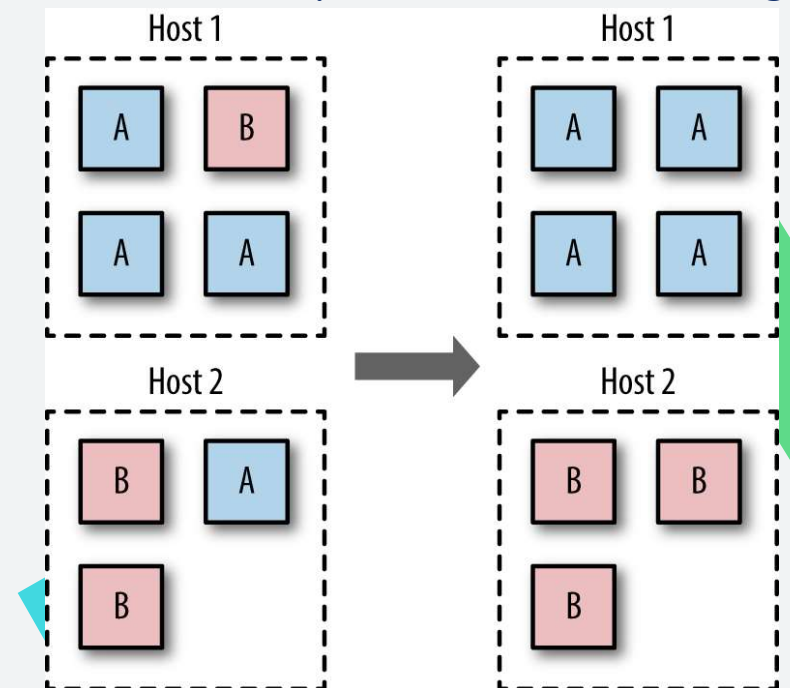
- The defense for our system should consist of multiple layers.
- For example, our containers will most likely run in VMs, so that if a container-breakout occurs, another level of defense can prevent the attacker from getting to the host or containers belonging to other users.
- Monitoring systems should be in place to alert admins in case of unusual behavior.
- Firewalls may restrict network access and limits the external attack surface.

## Least Privilege

- Each process and container should run with the minimum set of access rights and resources it needs to perform its function.
- The main benefit of this approach is that if one container is compromised, the attacker should still be severely limited in being able to access further data or resources.
- Regarding least privilege, you can take many steps to reduce the capabilities of containers, such as:
  - ❖ Ensure that processes in containers do not run as root so that exploiting a vulnerability present in a process does not give the attacker root access.
  - ❖ Run filesystems as read-only so that attackers cannot overwrite data or save malicious scripts to file.
  - ❖ Cut down on the kernel calls a container can make to reduce the potential attack surface.
  - ❖ Limit the resources a container can use to avoid DoS attacks where a compromised container or application consumes enough resources (such as memory or CPU) to bring the host to a halt.

## Segregate Containers by Host

- If we have a multitenancy setup where we are running containers for multiple users (whether these are internal users in our organization or external customers), ensure each user is placed on a separate Docker host.
- This is less efficient than sharing hosts between users and will result in a higher number of VMs and/or machines than reusing hosts but is important for security.
- The main reason is to prevent container breakouts resulting in a user gaining access to another user's containers or data.
- If a container breakout occurs, the attacker will still be on a separate VM or machine and unable to easily access containers belonging to other users.



## Applying Updates

- The ability to quickly apply updates to a running system is critical to maintaining security, especially when vulnerabilities are disclosed in common utilities and frameworks.
- The process of updating a containerized system roughly involves the following steps:
  - ❖ Identify images that require updating. This includes both base images and any dependent images.
  - ❖ Get or create an updated version of each base image. Push this version to your registry or download site.
  - ❖ For each dependent image, run docker build with the --no-cache argument
  - ❖ On each Docker host, run docker pull to ensure that it has up-to-date images.
  - ❖ Restart the containers on each Docker host.
  - ❖ Once you've determined that everything is functioning correctly, remove the old images from the hosts. If you can, also remove them from your registry

### Avoid Unsupported Drivers

- Despite its youth, Docker has already gone through several stages of development, and various features have been deprecated or unmaintained.
  - ❖ Relying on such features is a security risk, as they will not be receiving the same attention and updates as other parts of Docker.
- The same goes for drivers and extensions depended on by Docker.
  - ❖ In particular, do not use the legacy LXC execution driver.
  - ❖ By default, this is turned off, but you should check that your daemon isn't running with the -e lxc argument.
- Storage drivers are another major area of development and change.
  - ❖ Earlier, Docker is moving from AUFS to Overlay as the preferred storage driver.
  - ❖ The AUFS driver is being taken out of the kernel and is no longer under development.
  - ❖ Users of AUFS are encouraged to move to Overlay in the near future

## Image Provenance

- To safely use images, we need to have guarantees about their provenance: where they came from and who created them.
  - ❖ We need to be sure that we are getting exactly the same image the original developer tested and that no one has tampered with it, either during storage or transit.
  - ❖ If we can't verify this, the image may have become corrupted or—much worse—replaced with something malicious.
- Provenance is far from a new problem in computing. The primary tool in establishing the provenance of software or data is the secure hash.
  - ❖ A secure hash is something like a fingerprint for data—it is a small string that is unique to the given data.
  - ❖ Any changes to the data will result in the hash changing.
  - ❖ Several algorithms are available for calculating secure hashes, with varying degrees of complexity and guarantees of the uniqueness of the hash.
  - ❖ The most common algorithms are SHA (which has several variants) and MD5 (which has fundamental problems and should be avoided).



### Docker Digests

- Secure hashes are known as digests in Docker parlance. A digest is a SHA256 hash of a filesystem layer or manifest, where a manifest is metadata file describing the constituent parts of a Docker image.
- As the manifest contains a list of all the image's layers identified by digest, if you can verify that the manifest hasn't been tampered with, you can safely download and trust all the layers, even over untrustworthy channels (e.g., HTTP).

### Docker Content Trust

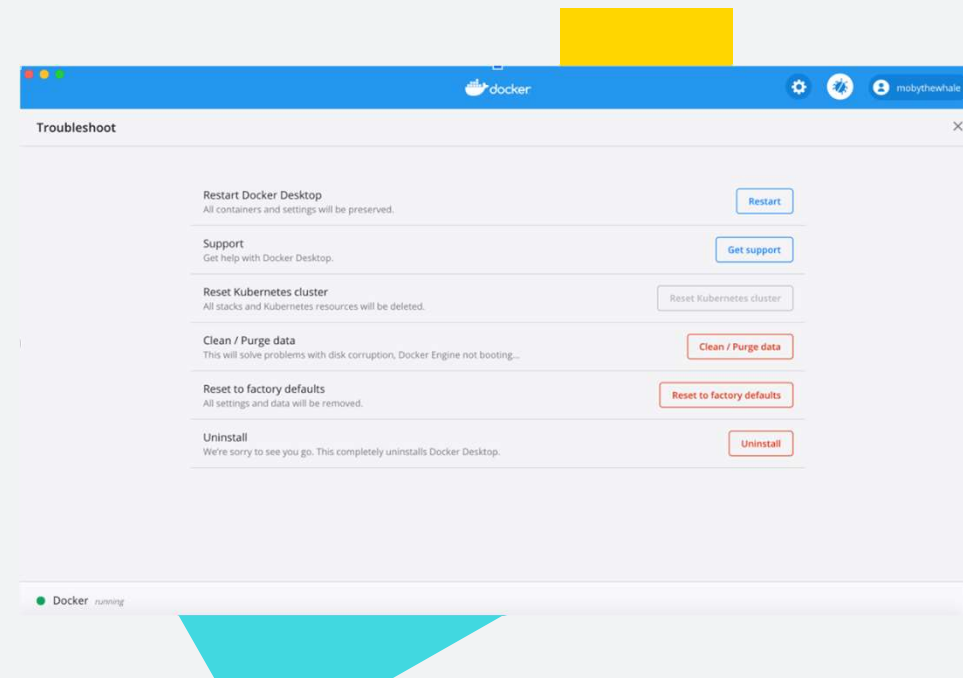
- Docker introduced content trust in 1.8. This is Docker's mechanism for allowing publishers to sign their content, completing the trusted distribution mechanism.
- When a user pulls an image from a repository, she receives a certificate that includes the publisher's public key, allowing her to verify that the image came from the publisher.
- When content trust is enabled, the Docker engine will only operate on images that have been signed and will refuse to run any images whose signatures or digests do not match.

## Security and Isolation : Security Tips

- **Set a User** : Never run production applications as root inside the container.
- **Limit Container Networking** : A container should open only the ports it needs to use in production, and these ports should be accessible only to the other containers that need them.
- **Remove Setuid/Setgid Binaries** : If we can disable or remove such binaries, we stop any chance of them being used for privilege escalation attacks.
- **Limit Memory** : This will protect against DoS attacks and applications with memory leaks that slowly consume all the memory on the host.
- **Limit CPU** : If an attacker can get one container, or one group of containers, to start using all the CPU on the host, that attacker will be able to starve out any other containers on the host, resulting in a DoS attack.
- **Limit Restarts** : If a container is constantly dying and restarting, it will waste a large amount of system time and resources. And may leads to causing a DoS.
- **Limit Filesystems** : Provide write access only to files which are mandatory
- **Limit Capabilities** : The Linux kernel defines sets of privileges, that can be assigned to processes to provide them with greater access to the system.
- **Apply Resource Limits (ulimits)** : Limiting the number of child processes that can be forked and the number of open file descriptors allowed.

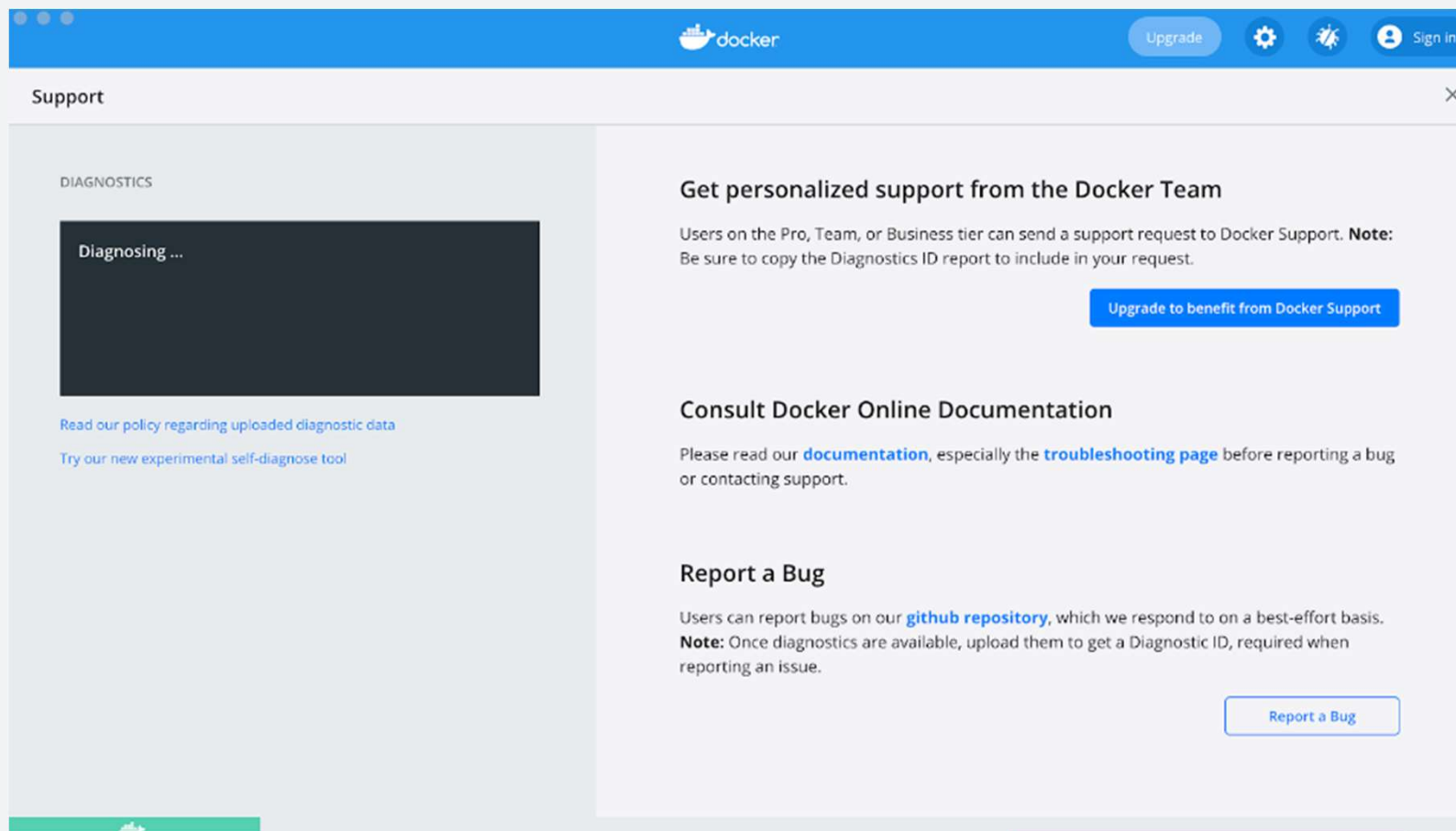
# Troubleshooting

- It contains information on how to diagnose and troubleshoot Docker Desktop issues, request Docker Desktop support, send logs and communicate with the Docker Desktop team, use our forums and Success Center, browse and log issues on GitHub, and find workarounds for known problems.
- Troubleshoot
- Choose Docker → Troubleshoot from the menu bar to see the troubleshoot options.
- ❖ **Restart Docker Desktop:** Restart Docker Desktop.
- ❖ **Support:** Send a support request (Paid subscribers) and do the diagnose any issues in Docker Desktop (For others).
- ❖ **Reset Kubernetes cluster:** To delete all stacks and Kubernetes resources.
- ❖ **Clean / Purge data:** To delete container and image data. Choose whether you'd like to delete data from Hyper-V, WSL 2, or Windows Containers and then click Delete to confirm.
- ❖ **Reset to factory defaults:** To reset all options on Docker Desktop to their initial state, the same as when Docker Desktop was first installed.



## Diagnose and feedback

- Choose Docker → Troubleshoot from the menu. Sign into Docker Desktop. In addition, ensure you are signed into your Docker account.
- Click Get support. This opens the in-app Support page and starts collecting the diagnostics.
- When the diagnostics collection process is complete, click Upload to get a **Diagnostic ID**. After getting this, copy the ID.
- If you have a paid Docker subscription, click Contact Support. This opens the Docker Desktop support form. Fill in the information required and add the ID you copied earlier to the Diagnostics ID field. Click Submit to request Docker Desktop support.
- If you don't have a paid Docker subscription, click Upgrade to benefit from Docker Support to upgrade your existing account. Alternatively, click Report a Bug to open a new Docker Desktop issue on GitHub. This opens Docker Desktop for Windows on GitHub in your web browser in a 'New issue' template. Complete the information required and ensure you add the diagnostic ID you copied earlier. Click submit new issue to create a new issue.



The screenshot shows the Docker Support page. At the top is a blue navigation bar with the Docker logo, an 'Upgrade' button, and icons for settings, a bug report, and a user profile with a 'Sign in' button. Below the navigation bar is a 'Support' header with a close button. The main content area is divided into two columns. The left column, titled 'DIAGNOSTICS', contains a dark grey box with the text 'Diagnosing ...', a link to 'Read our policy regarding uploaded diagnostic data', and a link to 'Try our new experimental self-diagnose tool'. The right column contains three sections: 'Get personalized support from the Docker Team' with a blue 'Upgrade to benefit from Docker Support' button; 'Consult Docker Online Documentation' with a link to the 'documentation' and a link to the 'troubleshooting page'; and 'Report a Bug' with a blue 'Report a Bug' button.

Support

DIAGNOSTICS

Diagnosing ...

[Read our policy regarding uploaded diagnostic data](#)

[Try our new experimental self-diagnose tool](#)

Get personalized support from the Docker Team

Users on the Pro, Team, or Business tier can send a support request to Docker Support. **Note:** Be sure to copy the Diagnostics ID report to include in your request.

[Upgrade to benefit from Docker Support](#)

Consult Docker Online Documentation

Please read our [documentation](#), especially the [troubleshooting page](#) before reporting a bug or contacting support.

Report a Bug

Users can report bugs on our [github repository](#), which we respond to on a best-effort basis. **Note:** Once diagnostics are available, upload them to get a Diagnostic ID, required when reporting an issue.

[Report a Bug](#)

## Diagnosing from the terminal

- On occasions it is useful to run the diagnostics yourself, for instance if Docker Desktop for Windows cannot start.
- To create and upload diagnostics in Powershell, run:  
`PS C:\> & "C:\Program Files\Docker\Docker\resources\com.docker.diagnose.exe" gather -upload`
- After the diagnostics have finished, you should have the following output, containing your diagnostic ID:
  - ❖ **Diagnostics Bundle:** C:\Users\User\AppData\Local\Temp\CD6CF862-9CBD-4007-9C2F-5FBE0572BBC2\20180720152545.zip
  - ❖ **Diagnostics ID:** CD6CF862-9CBD-4007-9C2F-5FBE0572BBC2/20180720152545 (uploaded)

## Self-diagnose tool

- Docker Desktop contains a self-diagnose tool which helps you to identify some common problems.  
`PS C:\> & "C:\Program Files\Docker\Docker\resources\com.docker.diagnose.exe" check`
- The tool runs a suite of checks and displays **PASS** or **FAIL** next to each check. If there are any failures, it highlights the most relevant at the end.

## Troubleshooting topics

- **Make sure certificates are set up correctly :**
  - ❖ Docker Desktop ignores certificates listed under insecure registries, and does not send client certificates to them.
- **Permissions errors on data directories for shared volumes**
  - ❖ When sharing files from Windows, Docker Desktop sets permissions on shared volumes to a default value of 777 (read, write, execute permissions for user and for group) and this default permissions are not configurable.
  - ❖ If you are working with applications that require permissions different from the shared volume defaults at container runtime, you need to either use non-host-mounted volumes or find a way to make the applications work with the default file permissions.
- **Volume mounting requires shared folders for Linux containers**
  - ❖ If we are using mounted volumes and get runtime errors indicating an application file is not found, access is denied to a volume mount, or a service cannot start, such as when using Docker Compose, we must enable shared folders.

- **Avoid unexpected syntax errors, use Unix style line endings for files in containers**
  - ❖ Any file destined to run inside a container must use Unix style `\n` line endings. This includes files referenced at the command line for builds and in RUN commands in Docker files.
- **Path conversion on Windows**
  - ❖ On Linux, the system takes care of mounting a path to another path.

```
$ docker run --rm -ti -v /home/user/work:/work alpine
```

- ❖ However, on Windows, you must update the source path. For example, if you are using the legacy Windows shell

```
$ docker run --rm -ti -v C:\Users\user\work:/work alpine
```

- ❖ This starts the container and ensures the volume becomes usable. This is possible because Docker Desktop detects the Windows-style path and provides the appropriate conversion to mount the directory.



- A good monitoring solution should show briefly the health of the system and give advance warning if resources are running low (e.g., disk space, CPU, memory).
- We also want to be alerted should things start going wrong (e.g., if requests start taking several seconds or more to process)

### Docker Tools

- Docker comes with a basic CLI tool, `docker stats`, that returns a live stream of resource usage.
- The command takes the name of one or more containers and prints various statistics for them, in much the same way as the Unix application `top`.

```
$ docker stats logging_logspout_1
CONTAINER          CPU %   MEM USAGE/LIMIT   MEM %   NET I/O
logging_logspout_1  0.13%   1.696 MB/2.099 GB  0.08%   4.06 kB/9.479 kB
```

- The stats cover CPU and memory usage as well as network utilization.
- Docker API that can be used to get such data programmatically.
- This API does indeed exist, and you can call the endpoint at `/containers/<id>/stats` to get a stream of various statistics on the container, with more detail than the CLI.

### Logstash

- While Logstash is very much a logging tool, it's worth pointing out that you can already achieve a level of monitoring with Logstash, and that the logs themselves are an important metric to monitor.
- For example, you could check nginx status codes and automatically email or message an alert upon receiving a high volume of 500s
- Logstash also has output modules for many common monitoring solutions, including Nagios, Ganglia, and Graphite.

### cAdvisor

- cAdvisor (a contraction of Container Advisor) from Google is the most commonly used Docker monitoring tool. It provides a graphical overview of the resource usage and performance metrics of containers running on the host.
- As cAdvisor is available as a container itself, we can get it up and running in a flash.



### Cluster Solutions

- cAdvisor is great but is a per-host solution. If you're running a large system, you will want to get statistics on containers across all hosts as well on the hosts themselves.
- You will want to get stats on how groups of containers are doing, representing both subsystems as well as slices of functionality across instances.
- For example, you may want to look at the memory usage of all your nginx containers, or the CPU usage of a set of containers running a data analysis task.
- Google has developed a cluster monitoring solution built on top of cAdvisor called **Heapster**.
- Heapster is marked **deprecated** as of **Kubernetes 1.11**.
- Users are encouraged to use metrics-server instead, potentially supplemented by a third-party monitoring solution, such as Prometheus.

### Prometheus

- Prometheus is a free software application used for event monitoring and alerting.
- It records real-time metrics in a time series database built using a HTTP pull model, with flexible queries and real-time alerting.
- Applications are expected to expose metrics themselves, which are then pulled by the Prometheus server rather than sending metrics directly to Prometheus.
- The Prometheus UI can be used to query and graph data interactively, and the separate Prom Dash can be used to save graphs, charts, and gauges to a dashboard.
- Prometheus also has an Alert manager component that can aggregate and inhibit alerts and forward to notification services, such as email, and specialist services, such as PagerDuty and Pushover

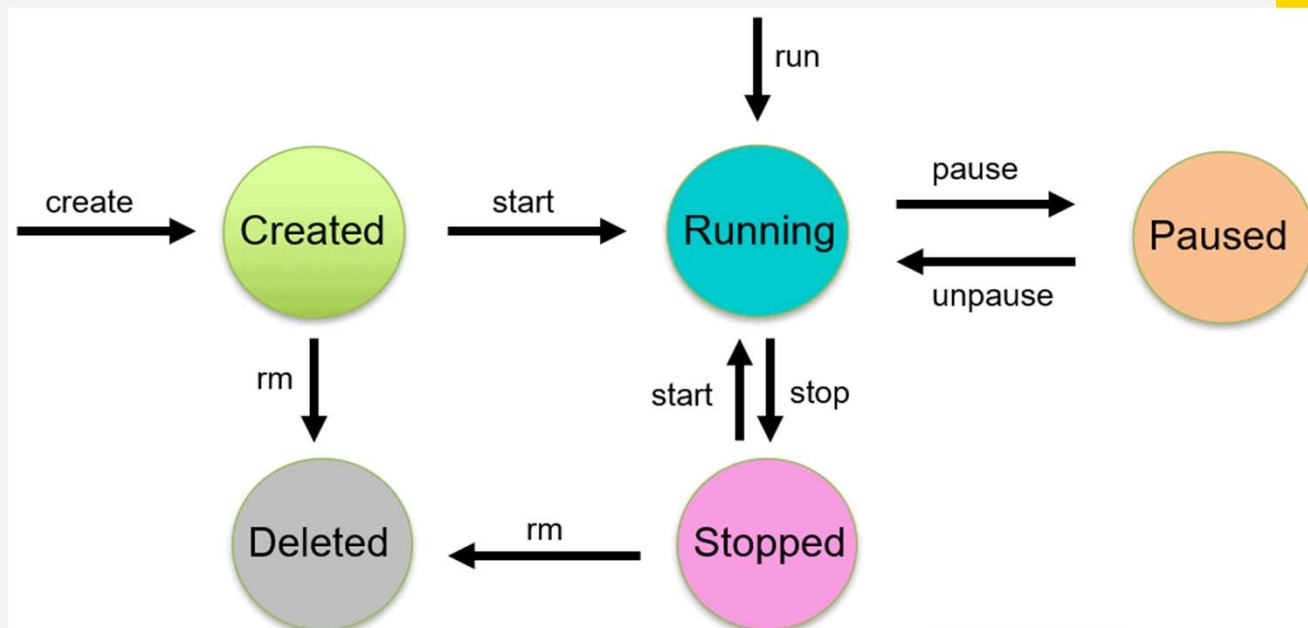


# Controlling running Containers

## Controlling Docker Containers

### ➤ Stages

- ❖ **Created:** A container that has been created but not started
- ❖ **Running:** A container running with all its processes
- ❖ **Paused:** A container whose processes have been paused
- ❖ **Stopped:** A container whose processes have been stopped
- ❖ **Deleted:** A container in a dead state



## ➤ Create Containers

- ❖ Using the docker create command will create a new Docker container with the specified docker image.
- ❖ \$ docker create --name <container name> <image name>

```
[root@localhost ~]# docker create --name c1 ubuntu  
3c821473291074944366f070e38ee207fa45fa95a0dcf7ffe27291553852c4ea
```

## ➤ Start Container

- ❖ To start a stopped container, we can use the docker start command.
- ❖ \$ docker start <container name>

```
[root@localhost ~]# docker start c1  
c1
```

## ➤ Run Container

- ❖ The docker run command will do the work of both “docker create” and “docker start” command. This command will create a new container and run the image in the newly created container..
- ❖ \$ docker run -it --name <container name> <image name>

```
[root@localhost ~]# docker run -it --name c3 ubuntu  
root@ba51292cce81:/# [root@localhost ~]#
```

# Controlling running Containers

## ➤ Pause Container

- ❖ If we want to pause the processes running inside the container, we can use the “docker pause” command.
- ❖ \$ docker pause <container name>

```
[root@localhost ~]# docker pause c3  
c3
```

## ➤ Unpause Container

- ❖ To unpause the container, use “docker unpause” command.
- ❖ \$ docker unpause <container name>

```
[root@localhost ~]# docker unpause c3  
c3
```

## ➤ Stop Container

- ❖ Stopping a running Container means to stop all the processes running in that Container. Stopping does not mean killing or ending the process.
- ❖ \$ docker stop <container name>

```
[root@localhost ~]# docker stop c1  
c1
```

# Controlling running Containers

## ➤ Kill Container

- ❖ We can kill one or more running containers.
- ❖ \$ docker kill <container name>

```
[root@localhost ~]# docker kill c3  
c3
```

## ➤ Delete/Remove Container

- ❖ Removing or deleting the container means destroying all the processes running inside the container and then deleting the Container. It's preferred to destroy the container, only if present in the stopped state instead of forcefully destroying the running container.
- ❖ \$ docker rm <container name>

```
[root@localhost ~]# docker rm c3  
Error response from daemon: You cannot remove a running container ba51292cce814654483f0640f137f4bd3e6d7543830a098f3fb6e6d86634865e. Stop the container before attempting removal or force remove
```

```
[root@localhost ~]# docker stop c3  
c3  
[root@localhost ~]# docker rm c3  
c3
```



### Controlling Security

- Security is of prime importance when it comes to deciding whether to invest in a technology, especially when that technology has implications on the infrastructure and workflow.
- Docker containers are mostly secure, and since Docker doesn't interfere with other systems, you can use additional security measures to harden the security around the docker daemon.
- It is better to run the docker daemon in a dedicated host and run other services as containers.
- **Kernel namespaces**
  - ❖ Namespaces provide sandboxing to containers. When a container is started, Docker creates a set of namespaces and cgroups for the container.
  - ❖ Thus, a container that belongs to a particular namespace can't see or affect the behavior of another container that belongs to other namespaces or the host.
  - ❖ The namespace feature is modeled after OpenVZ, which is an operating system level virtualization technology based on the Linux kernel and operating system.

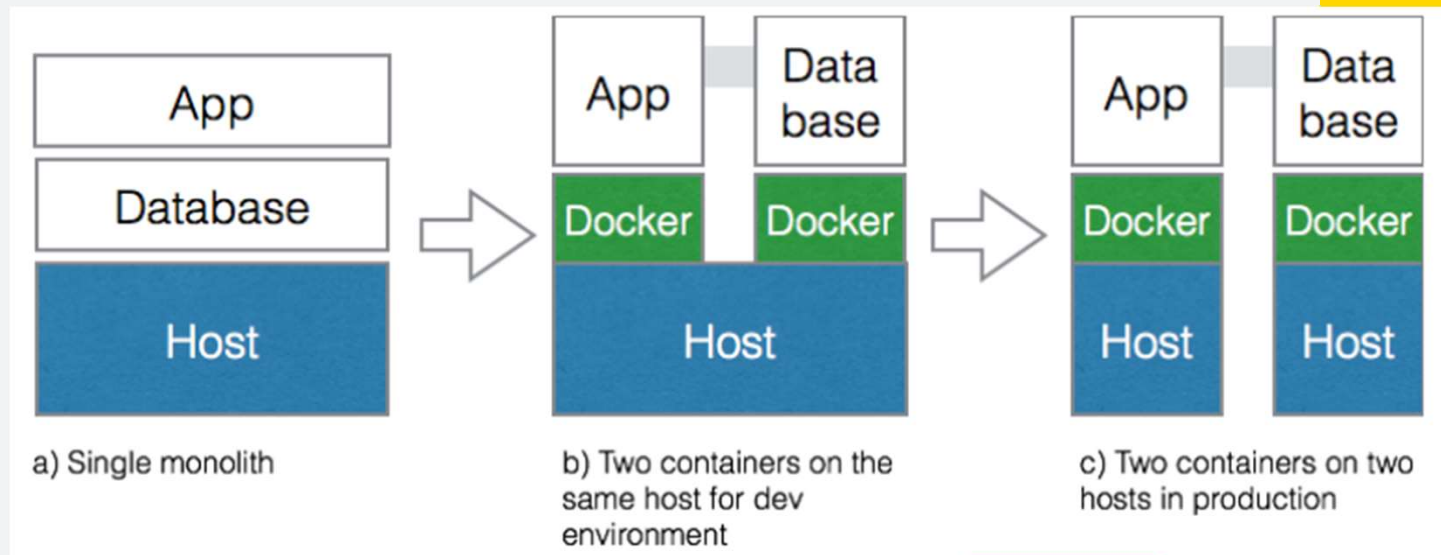
## ➤ Control groups

- ❖ Control groups provide resource management features. Although this has nothing to do with privileges, it is relevant to security because of its potential to act as the first line of defense against denial-of-service (DoS) attacks.
- ❖ Control groups have been around for quite some time as well, so can be considered safe for production use.
- ❖ Control Groups provide a mechanism for aggregating/partitioning sets of tasks, and all their future children, into hierarchical groups with specialized behavior.
- ❖ **cgroup** associates a set of tasks with a set of parameters for one or more subsystems.
- ❖ **Subsystem** is a module that makes use of the task grouping facilities provided by cgroups to treat groups of tasks in particular ways.
- ❖ A subsystem is typically a "**resource controller**" that schedules a resource or applies per-cgroup limits, but it may be anything that wants to act on a group of processes.  
e.g., a virtualization subsystem.
- ❖ **Hierarchy** is a set of cgroups arranged in a tree, such that every task in the system is in exactly one of the cgroups in the hierarchy, and a set of subsystems; each subsystem has system-specific state attached to each cgroup in the hierarchy.

- Containers allow software developers to package everything an application needs into its own, self-contained bundle. Each bundle, or container, includes the application itself, plus any dependencies, libraries and configuration files it needs to run.
- Containers let developers create applications using small services accessible through an API rather than a monolithic application. When using this type of microservices architecture, developers can easily make a small change to an application and push it out immediately--without affecting other microservices.
- Containers offer many advantages to the business environment.
  - ❖ **Speed**
    - ❑ Spinning up new environments can be a matter of seconds instead of minutes or hours as with other virtualization technologies.
    - ❑ Once an image exists on the host you can spin up a new container ready to take requests in less than 1 second in many cases.
  - ❖ **Density:**
    - ❑ Most containers are megabytes in size rather than gigabytes. With their smaller footprint, containers allow for as much greater density than VMs.

## ❖ Flexibility & Consistency

- ❑ Containers offer more flexibility in both development and deployment.
- ❑ For instance, the same definition that creates the Docker container on a dev environment could be deployed in separate servers in production.
- ❑ They could even be clustered. It can make the definition and management more convenient.
- ❑ Environments can be managed by environment variables instead of juggling config files with tools like Puppet or Capistrano.



### ❖ Cloud-readiness

- ❑ Containers can be deployed to any cloud (private, public, virtual or physical) which opens up the possibility of using the same containerized application in hybrid and multi-cloud environments.

### ❖ Agility

- ❑ Being able to define our environments in code means we can better leverage our existing source control tools (e.g., Git, SVN) to version your environments just as we would our code.
- ❑ These environments as mentioned earlier can move with you from development through production.
- ❑ This means your continuous integration tools (or continuous deployment) can also leverage these container definitions.
- ❑ CI / CD is changing the way companies build and scale their applications (think of Jenkins, CircleCI, & Wercker).
- ❑ In our projects we have seen using tools like Docker and CircleCI you can have clean environments spun up and full test suites ran in just a few minutes. This would take 10s of minutes not long ago.

### ❖ Easier to scale

- ❑ We can better leverage our existing source control tools (e.g., Git, SVN) to version your environments just as we would our code.
- ❑ These environments can be used to move from development through production easily.
- ❑ This means your continuous integration tools (or continuous deployment) can also leverage these container definitions.
- ❑ CI / CD is changing the way companies build and scale their applications (think of Jenkins, CircleCI, & Wercker).
- ❑ In our projects we have seen using tools like Docker and CircleCI you can have clean environments spun up and full test suites ran in just a few minutes. This would take 10s of minutes not long ago.

### ❖ Blast radius

- ❑ As software developers incorporate and release changes more quickly, they look for ways to control potential damage from bugs. A problem in a container is less likely to cause widespread damage and is easier to isolate and repair than a problem in an entire VM or physical server.

### ❖ Cost savings

- ❑ Every VM requires a hypervisor and an operating system (OS), which can involve licensing and fees.
- ❑ Containers run side by side, sharing the OS kernel of a single physical machine while holding isolated copies of specific library versions they depend on.
- ❑ Architecturally, VMs tend to contain more components and require greater allocation of resources. Containers, on the other hand, tend to hold a specific component or a single micro-service, minimizing resource overhead.

### ❖ Microservices

- ❑ The notion of running dozens of independent APIs and apps in any practical and managed way would be out of reach of most mid-sized companies.
- ❑ With the right architectures, automation, and containers microservices become practical for a whole new market of businesses.

## Reference

<https://blogs.perficient.com/2019/08/21/the-benefits-of-containers-for-businesses>

