A container is a small package of code that comes with all of its dependencies so you can quickly deploy it and easily move it from one environment to another. So, when is the best time to use containers and what security benefits do they provide?

In our first project we deployed all services we used as a container including ansible, DVWA, and ELK. This allowed us to get all of these services up and running as well as customizing them with options such as ports quite easily using an ansible playbook. This is the ideal use of containers to allow fast deployment, easy customization as well as the control to easily turn a container off when we are done using it.

Security wise, containers are a much more ideal solution than running services manually as each docker container runs on it’s own network stack so that the containers cannot default communicate with IP traffic. This can of course be added by linking different docker containers as well as mapping external ports to one of the docker ports. Docker containers also have built in control groups that allow you to view the metrics of the container such as resource usage, but more importantly it limits the system resources that a docker container can use so that one single container can’t crash a system by using up all of the resources therefore adding a layer of defense to a denial-of-service attack. The docker daemon also gives permissions to containers dynamically so they only have what they need to work and handles processes that are normally run as a root user outside of the container itself so that it doesn’t need root access, for example SSH and cron are handled by the external docker shell and not given to the individual containers. This means that if an attacker were to manage to escalate themselves to a root user within a container it would not have nearly the same power as the real root user on the underlying system.

In our first project we initially setup docker manually on the jump box vm and got ansible to be able to use ansible playbooks to establish the rest of our containers. Once we had access to ansible playbooks we could create scripts that are reusable to install docker and dependencies as well as download and run docker containers. After ensuring we had the proper container name, we could add it to a playbook with all the configurations we need such as proper port mappings and setting it to restart if it is closed. After running a playbook, we can use `sudo docker ps` to ensure that docker properly installed and the container that we want up is running.

If we wanted to setup ELK and DVWA without containers we would have to manually install them as well as all dependencies that they need which will live on the main computer, configuration will also have to be done manually by editing the individual files that require them. Setting up services without a container does pose some advantages such as although containers are mor lightweight than an entire virtual machine they still can’t operate at the speeds they could running on the base system. Containers also keep data stored inside of the container and when the container is deleted you lose all the data inside of it and although volumes exist to keep data on the host system you will still lose configuration. Running anything with a GUI does not do well in a container since they weren’t made with that in mind. Running a service on the host system also allows you to receive updates much faster as you can install them as soon as they are deployed. Despite these drawbacks’ containers have many advantages such as their ability to be quickly deployed with minimal configuration as they come pre built with all their dependencies. They can also scale incredibly well with docker swarm or Kubernetes. Containers can also be used across multiple operating systems and doesn’t require OS specific installation. You can instantly update the service once an update releases and although you will need to wait until the docker image itself is updated it will be able to update itself.