Elk Technologies
https://elktech.org
Scott Stephenson

Module 6: Application Deployment and Security

DevNet Associate v1.0

Cisco | Networking Academy®
Mind Wide Open™

Module Objectives

- Module Title: Application Deployment and Security
- Module Objective: Use current technologies to deploy and secure applications and data in a cloud environment.
- It will comprise of the following sections:

Topic Title	Topic Objective
6.1 Understanding Deployment Choices with Different Models	Explain common cloud deployment models.
6.2 Creating and Deploying a Sample Application	Use container technology to deploy a simple application.
6.3 Continuous Integration/Continuous Deployment (CI/CD)	Explain the use of Continuous Integration/Continuous Deployment (CI/CD) in application deployment.
6.4 Networks for Application Development and Security	Explain the network technology required for application development in a cloud environment.
6.5 Securing Applications	Use common application security techniques to secure data.

6.1 Understanding Deployment Choices with Different Models

Introduction to Deployment Choices

- Developers need to do more than deliver application code: they need to concern themselves with how applications are deployed, secured, operated, monitored, scaled, and maintained.
- The physical and virtual infrastructure and platforms on which applications are being developed and deployed are quickly evolving.
- Developers are confronted with an expanding stack of platform options, which are all hosted on infrastructures and frameworks of increasing flexibility and complexity.

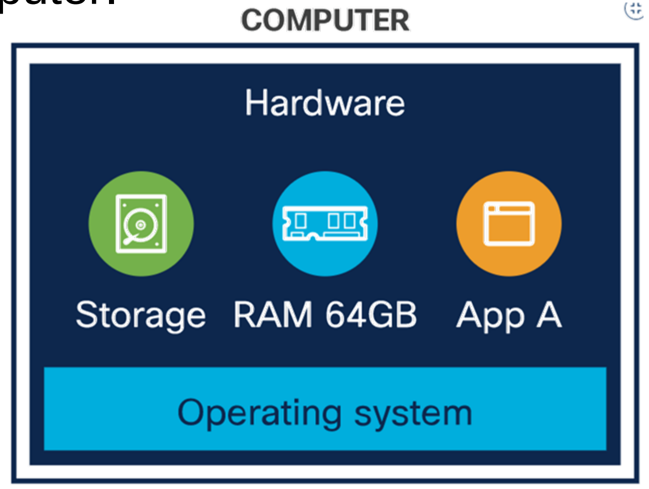
Deployment Environments

- A piece of code, before it reaches to the user, passes through a number of environments that leads to an increase in its quality and reliability. These environments are self-contained and mimic the ultimate environment in which the code will live.
- Typically, large organizations use a four-tier structure:

Development	Testing	Staging	Production
• This environment is used for coding. I • t is also used to manage fundamental aspects of the infrastructure, such as containers or cloud networking.	• This environment is used for testing the code. • It often includes automated tools such as Jenkins, Circled, or Travis CI. • It is often integrated with a version control system	• This environment is structured as close as possible to the actual production environment. • It is used for final acceptance testing in a realistic environment.	• This environment contains code that has been tested multiple times and is error free. • It is used for deploying the final code for the end user interaction.

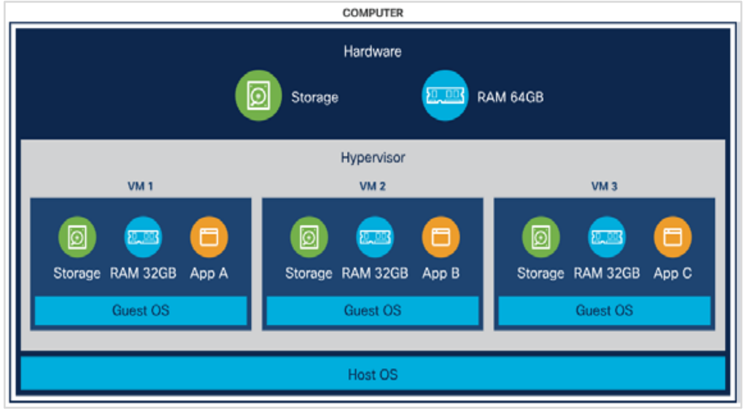
Deployment Models

- There are various deployment models that can be used to deploy a software. These include Bare Metal, Virtual Machines, Container-based Infrastructure and Serverless Computing.
- **Bare Metal**
  - A bare metal deployment is essentially deploying to an actual computer. It is used to install a software directly on the target computer.
  - In this method, software can directly access the operating system and the hardware.
  - It is useful for situations requiring access to specialized hardware, or for High Performance Computing (HPC) applications.
  - It is now used as infrastructure to host virtualization and cloud frameworks.



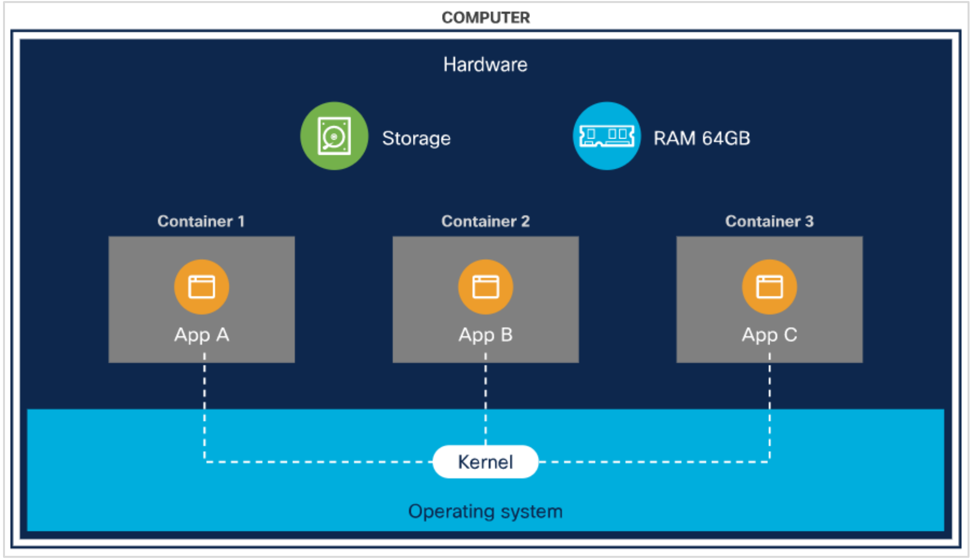
Deployment Models

- **Virtual Machines (VMs)**
  - Virtual machines share the resources of the host but is completely self-contained. It is like a computer within the computer and has its own memory, network interfaces, storage, and operating system.
  - Hypervisor is software that creates and manages VMs.
  - VMs run on top of a hypervisor that provides VMs with simulated hardware, or with controlled access to underlying physical hardware.



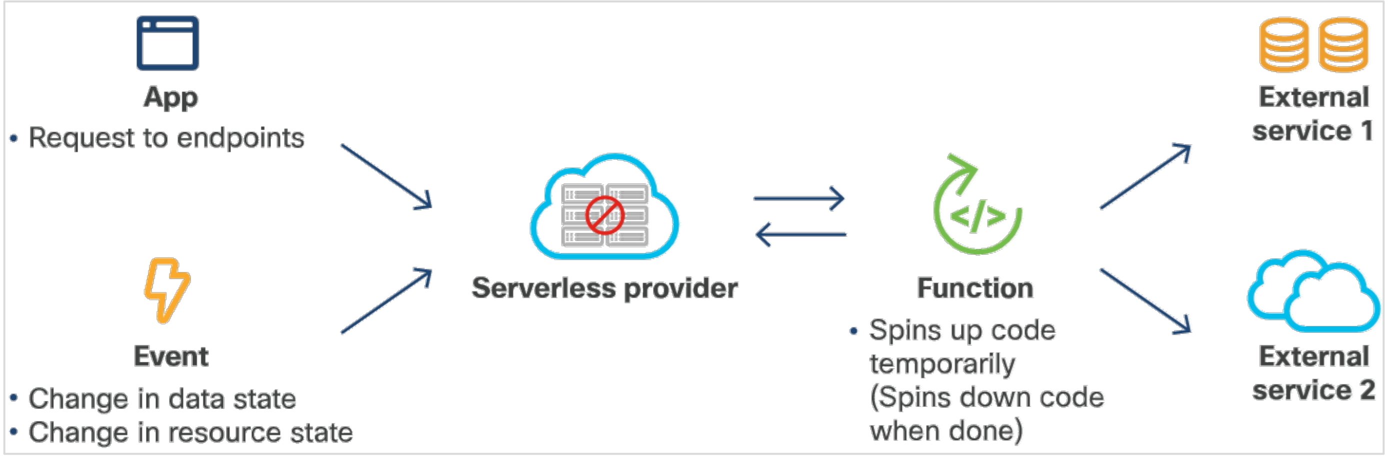
Deployment Models

- **Container-based infrastructure**
  - Containers were designed to provide the same benefits as VMs, such as workload isolation and the ability to run multiple workloads on a single machine but are designed to start up quickly.
  - Containers share resources of the host including the kernel.
  - A container shares the operating system of the host machine and uses container-specific binaries and libraries.



Deployment Models

- **Serverless Computing**
  - Serverless computing takes advantage of a modern trend towards applications that are built around services. Application makes a call to another program or workload to accomplish a particular task, to create an environment where applications are made available on an "as needed" basis.
  - Serverless computing takes responsibility for assigning resources away from the developer and only incurs costs when the application runs.



Deployment Models

- It works as follows:
  - Step 1. The developer creates an application.
  - Step 2. The developer deploys the application as a container, so that it can run easily in any appropriate environment.
  - Step 3. The developer deploys that container to a serverless computing provider. This deployment includes a specification of how long the function should remain inactive before it is spun down.
  - Step 4. When necessary, the application calls the function.
  - Step 5. The provider spins up an instance of the container, performs the needed task, and returns the result.

Types of Infrastructure

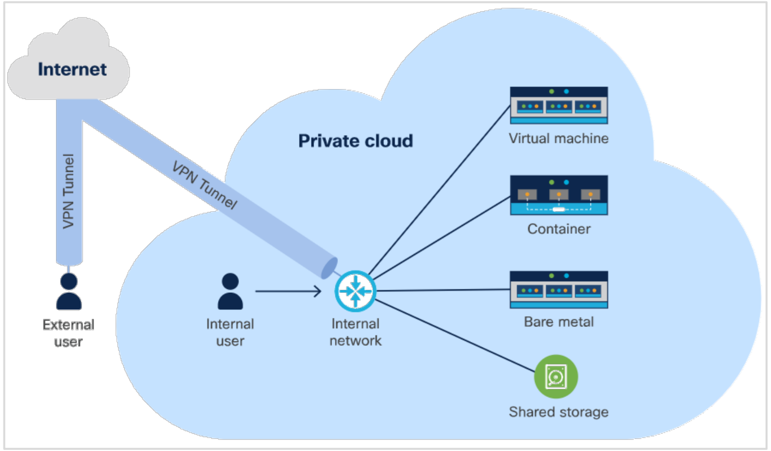
- In the early days of computers, infrastructure was pretty straightforward. Software ran on a single computer and networks could link multiple computers together.
- Now, infrastructure has become more complicated, with various options available for designing the infrastructure such as different types of clouds, and what each does and does not do well.

On-Premises

- On-Premises means any system that is literally within the confines of the building.
- On-Premises are the traditional data centers that house individual machines which are provisioned for applications, rather than clouds.
- These traditional data centers with servers dedicated to individual applications, or to VMs, which enable a single computer to act like multiple computers.
- Operating a traditional on-premises data center requires servers, storage devices, and network equipment to be ordered, received, assembled in racks, moved to a location, cabled for power and data. All this setup of infrastructure takes time and effort.
- Problems related to On-Premises can be solved by moving to a cloud-based solution.

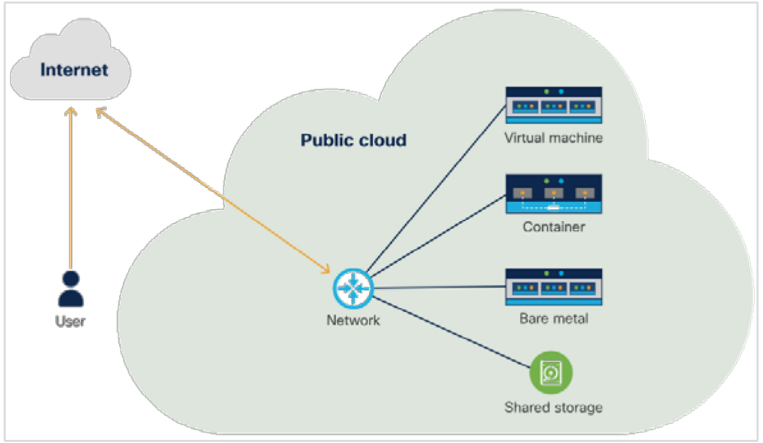
Private Cloud

- A cloud is a system that provides self-service provisioning for compute resources, networking, and storage.
- Private clouds are intended for a specific organization or entity.
- In a private cloud infrastructure, the organization controls all of the resources.
- In most cases, a private cloud is located in a data center and all resources that run on the hardware belong to the owner organization.
- The advantage of a private cloud is that one has complete control over where it is located.
- An operations team is required to manage the cloud and keep it running.



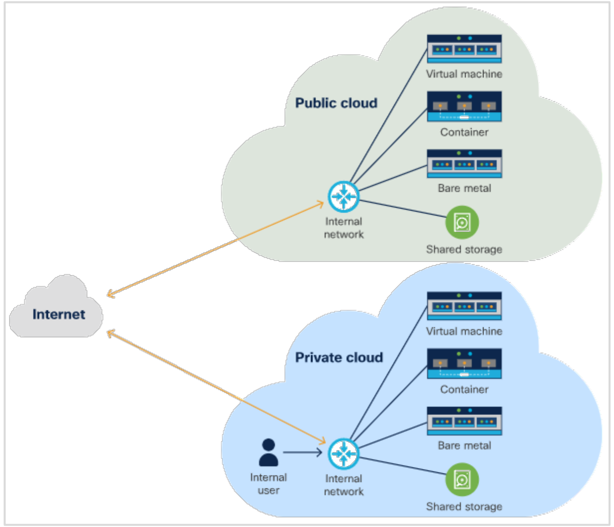
Public Cloud

- A public cloud is the same as a private cloud, but it is managed by a public cloud provider.
- Public cloud customers may share resources with other organizations. Alternatively, public cloud providers may provide customers with dedicated infrastructure.
- With a public cloud, the organization does not control the resources.
- A public cloud is helpful in scaling up virtually as long as the load requires and then scale down when traffic is slow.
- One disadvantage of public cloud is known as the "noisy neighbor" problem.



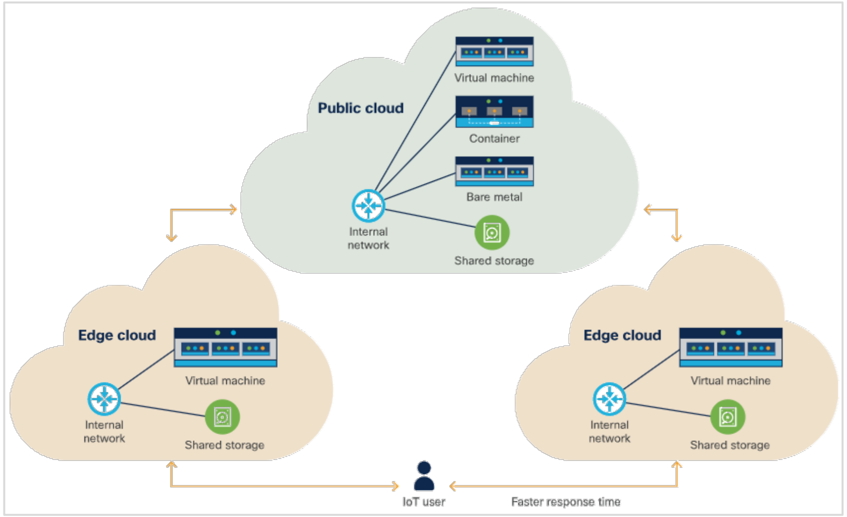
Hybrid Cloud

- Hybrid cloud is the combination of two different types of clouds.
- Hybrid cloud is used to bridge a private cloud and a public cloud within a single application.
- Hybrid cloud combines public and private cloud to provide additional resources and security where necessary.
- Hybrid cloud is distinguished by the use of more than one cloud within a single application.
- Container orchestrators have become very popular with companies employing hybrid-cloud deployments.



Edge Cloud

- Edge cloud is gaining popularity because of the growth of the Internet of Things (IoT).
- Edge cloud enables resources to be closer to where they are needed.
- Edge cloud computing comprises one or more central clouds that act as a hub for the edge clouds themselves.
- Hardware for the edge clouds is located as close as possible to the user.
- Edge cloud run on much smaller hardware so they may be more resource-constrained.



6.2 Creating and Deploying a Sample Application

What is Docker?

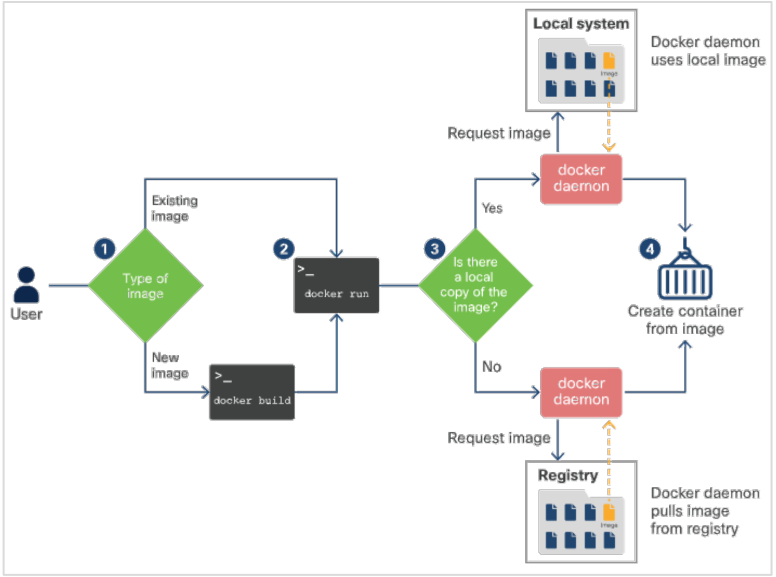
- The most popular way to containerize an application is to deploy it as a **Docker container**. A container is a way of encapsulating everything you need to run your application, so that it can easily be deployed in a variety of environments. Docker is a way of creating and running that container.
- A Docker image is a set of read-only files that have no state and contains source code, libraries, and other dependencies needed to run an application.
- A Docker container is the run-time instance of a Docker image.
- Creating a container involves pulling an image or a template from a repository, then using it to create a container.

What is Docker?

- Docker is a format that wraps a number of different technologies to create containers. These technologies are:
  - **Namespaces** - These isolate different parts of the running container.
  - **Control groups** - These `cgroups` are a standard Linux concept that enables the system to limit the resources, used by an application.
  - **Union File Systems** - These `UnionFS` are file systems that are built layer by layer, combining resources.

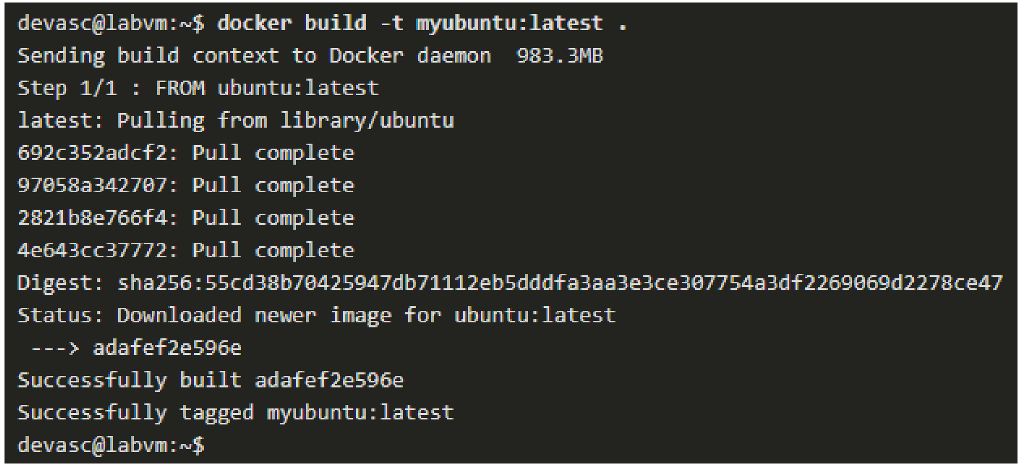
What is Docker?

- The workflow of creating a container is as follows:
  - Step 1: Either create a new image using `docker build` or pull a copy of an existing image from a registry using `docker pull`.
  - Step 2: Run a container based on the image using `docker run` or `docker container create`.
  - Step 3: The Docker daemon checks to see if it has a local copy of the image. If it does not, it pulls the image from the registry.
  - Step 4: The Docker daemon creates a container based on the image and, if docker run was used, logs into it and executes the requested command.



What is Dockerfile?

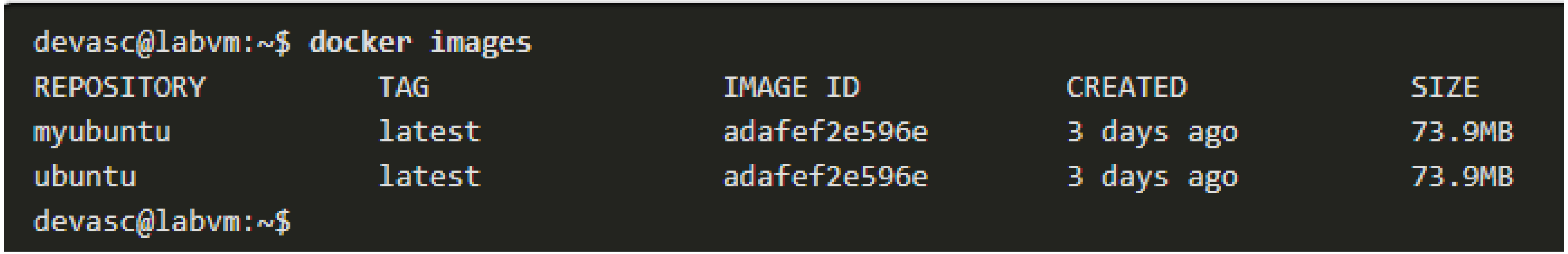
- **Dockerfile** is a simple text-file which is required to compile the code.
- It defines the steps that the docker build command takes to create an image that can be used to create the target container.
- Create a file named Dockerfile and save it in the current directory.
- Run the `docker build` command to build the image using a Dockerfile in the current directory (`.`) and give it a name of `myubuntu`.
- Finally, you can add a tag `:latest`
  - If you do not specify a tag, **latest** will be used by default?
- Steps to generate a Dockerfile that creates an Ubuntu container:



```
devasc@labvm:~$ docker build -t myubuntu:latest .
Sending build context to Docker daemon  983.3MB
Step 1/1 : FROM ubuntu:latest
latest: Pulling from library/ubuntu
692c352adcf2: Pull complete
97058a342707: Pull complete
2821b8e766f4: Pull complete
4e643cc37772: Pull complete
Digest: sha256:55cd38b70425947db71112eb5dddfa3aa3e3ce307754a3df2269069d2278ce47
Status: Downloaded newer image for ubuntu:latest
 ---> adafef2e596e
Successfully built adafef2e596e
Successfully tagged myubuntu:latest
devasc@labvm:~$
```

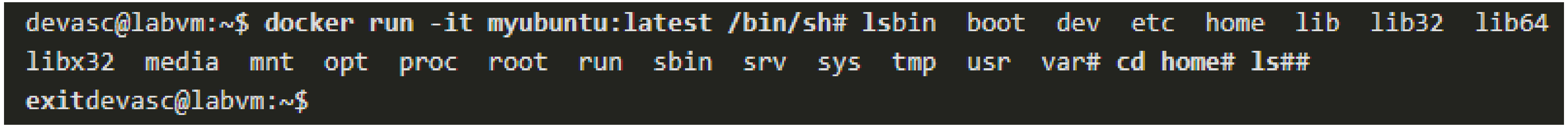
What is Dockerfile?

- Enter the command `docker images` to see your image in the list of images on the DEVASC VM.



```
devasc@labvm:~$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
myubuntu            latest              adafef2e596e        3 days ago          73.9MB
ubuntu              latest              adafef2e596e        3 days ago          73.9MB
devasc@labvm:~$
```

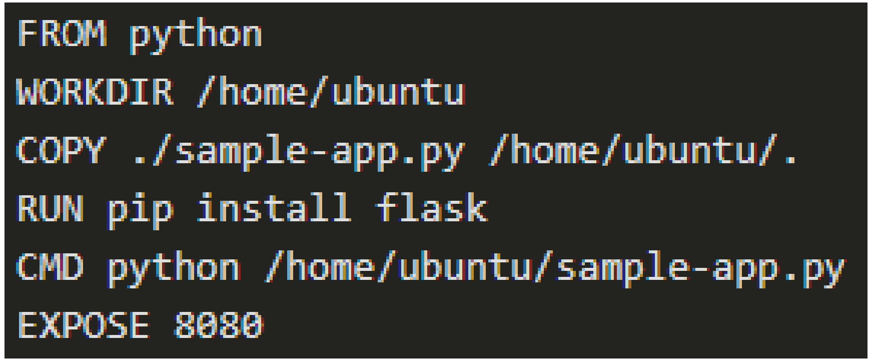
- Change to the home directory and enter `ls` to see that it is empty and ready for use.
- Enter `exit` to leave the Docker container and return to your DEVASC VM main operating system.



```
devasc@labvm:~$ docker run -it myubuntu:latest /bin/sh# lsbin   boot   dev   etc   home   lib   lib32   lib64
libx32   media   mnt   opt   proc   root   run   sbin   srv   sys   tmp   usr   var# cd home# ls##
exitdevasc@labvm:~$
```

Anatomy of a Dockerfile

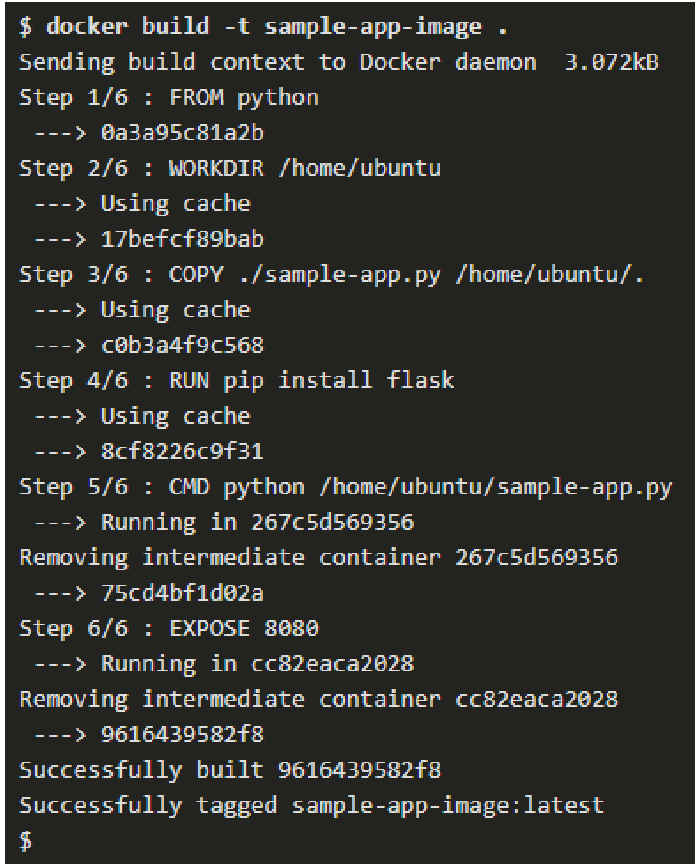
- Consider the following `Dockerfile` that containerizes a Python app and the explanation of the commands are as follows:
  - The `FROM` command installs Python in the Docker image.
  - The `WORKDIR` command tells Docker to use `/home/ubuntu` as the working directory.
  - The `COPY` command tells Docker to copy the file from Dockerfile’s current directory into `/home/ubuntu`.
  - The `RUN` command allows to directly run commands on the container.
  - The `CMD` command will start the server when the user run the actual container.
  - The `EXPOSE` command tells Docker that the user want to expose port 8080.



```
FROM python
WORKDIR /home/ubuntu
COPY ./sample-app.py /home/ubuntu/.
RUN pip install flask
CMD python /home/ubuntu/sample-app.py
EXPOSE 8080
```

Anatomy of a Dockerfile

- Docker takes advantage of what is stored in cache to speed up the process.
- The `docker build` command is used to build the image. In the given output, the image was previously built.
- The Docker goes through each step in the Dockerfile, starting with the base image, Python. If this image does not exist on the system, Docker pulls it from the registry. The default registry is Docker Hub.
- Between steps such as executing a command, Docker actually creates a new container and builds an intermediate image, a new layer, by saving that container.



```
$ docker build -t sample-app-image .
Sending build context to Docker daemon  3.072kB
Step 1/6 : FROM python
 ---> 0a3a95c81a2b
Step 2/6 : WORKDIR /home/ubuntu
 ---> Using cache
 ---> 17befcf89bab
Step 3/6 : COPY ./sample-app.py /home/ubuntu/.
 ---> Using cache
 ---> c0b3a4f9c568
Step 4/6 : RUN pip install flask
 ---> Using cache
 ---> 8cf8226c9f31
Step 5/6 : CMD python /home/ubuntu/sample-app.py
 ---> Running in 267c5d569356
Removing intermediate container 267c5d569356
 ---> 75cd4bf1d02a
Step 6/6 : EXPOSE 8080
 ---> Running in cc82eaca2028
Removing intermediate container cc82eaca2028
 ---> 9616439582f8
Successfully built 9616439582f8
Successfully tagged sample-app-image:latest
$
```

Start a Docker Container Locally

- After building the image using dockerfile, create a new container and do some work by entering the `docker run` command.
- The `-d` parameter is short for –detach and indicates that the image should run in the background.
- The `-P` parameter tells Docker to publish it on the port that was exposed.

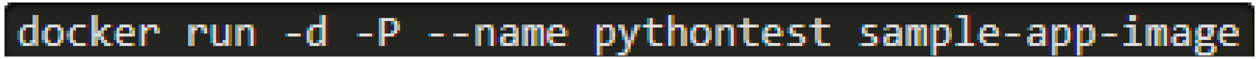
```
$ docker run -d -P sample-app-image
1688a2c34c9e7725c38e3d9262117f1124f54685841e97c3c5225af88e30bfc5
$
```

- Notice the container's listing processes:

```
$ docker ps
CONTAINER ID        IMAGE                   COMMAND                     CREATED             STATUS
PORTS                   NAMES
90edd03a9511        sample-app-image    "/bin/sh -c 'python …"   5 seconds ago       Up 3 seconds
0.0.0.0:32774->8080/tcp   jovial_sammet
$
```

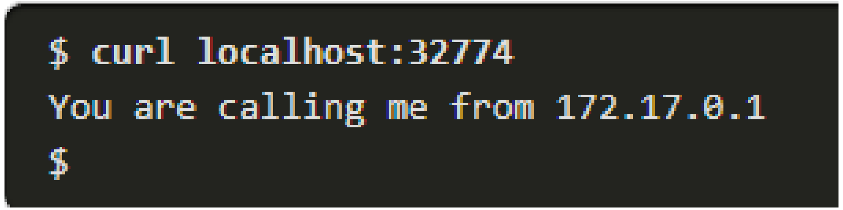
Start a Docker Container Locally

- Notice that Docker has assigned the container a name as `pythontest`. Naming is also done by with the `--name` option.



```
docker run -d -P --name pythontest sample-app-image
```

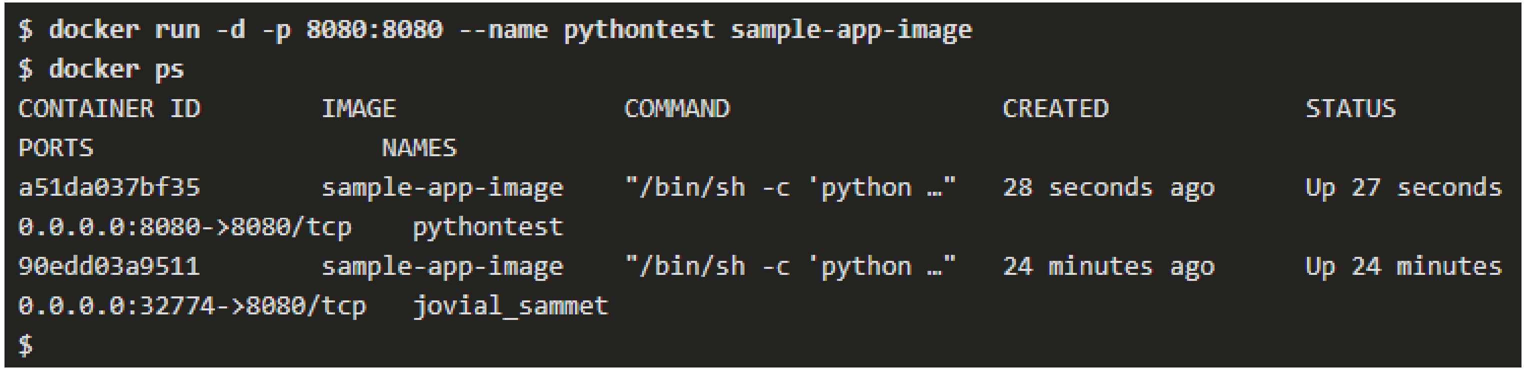
- Even though the container is listening on port 80, it is just an internal port. Docker has specified an external port as **32774**, which will forward to the internal port.
- This lets you run multiple containers which listen on the same port without having conflicts.
- To pull up the sample app website, use the public IP address for the host server and that port is used.



```
$ curl localhost:32774
You are calling me from 172.17.0.1
$
```

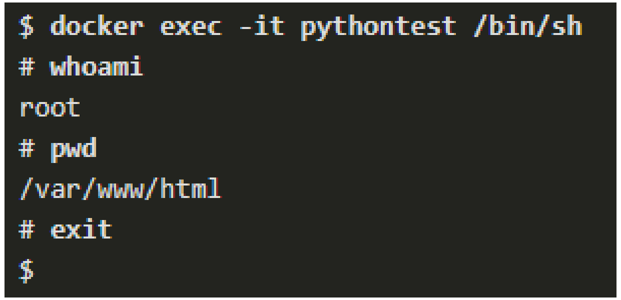
Start a Docker Container Locally

- Docker also allows to specify a particular port to forward, so that a more predictable system can be created.



```
$ docker run -d -p 8080:8080 --name pythontest sample-app-image
$ docker ps
CONTAINER ID        IMAGE                   COMMAND                   CREATED             STATUS
PORTS                     NAMES
a51da037bf35        sample-app-image    "/bin/sh -c 'python …"   28 seconds ago      Up 27 seconds
0.0.0.0:8080->8080/tcp    pythontest
90edd03a9511        sample-app-image    "/bin/sh -c 'python …"   24 minutes ago      Up 24 minutes
0.0.0.0:32774->8080/tcp   jovial_sammet
$
```

- When the container is running, logging into its activity can be executed using the `exec` command.



```
$ docker exec -it pythontest /bin/sh
# whoami
root
# pwd
/var/www/html
# exit
$
```

Start a Docker Container Locally

- To `stop` and remove `rm` a running container, call it by its name:

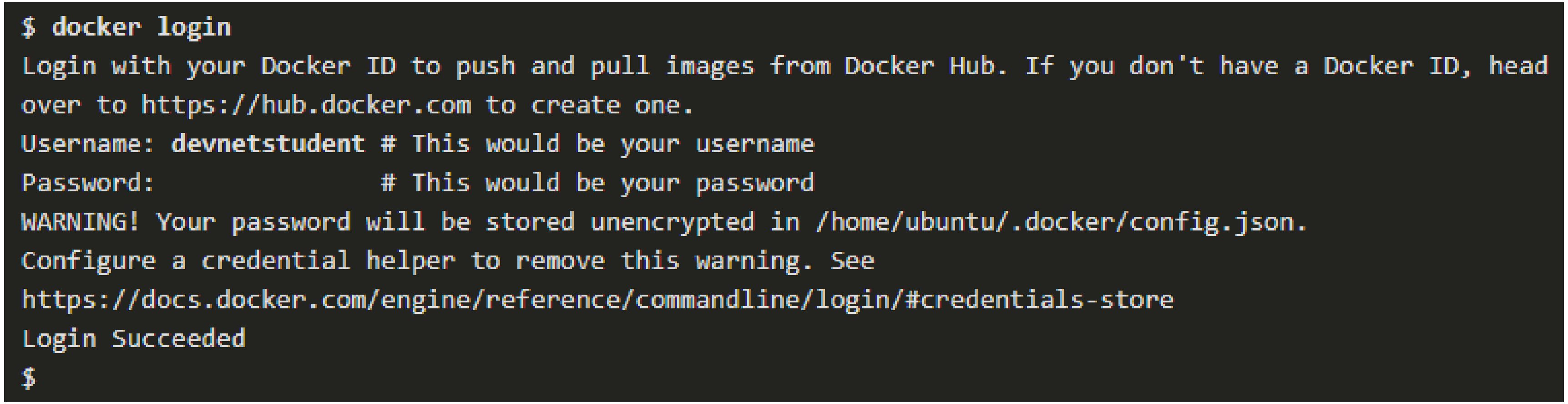
```
$ docker stop pythontest
pythontest
$ docker rm pythontest
pythontest
$
```

- Notice the running processes again and the running container has been removed.

```
$ docker ps
CONTAINER ID        IMAGE               COMMAND                   CREATED             STATUS
PORTS                     NAMES
90edd03a9511        sample-app-image    "/bin/sh -c 'python …"   25 minutes ago      Up 25 minutes
0.0.0.0:32774->8080/tcp   jovial_sammet
$
```

Save a Docker Image to a Registry

- To make the image available for users, store it in an image registry.
- By default, Docker uses the Docker Hub registry, but users can create their own registry too. To start the process:
- Log in to the registry.



```
$ docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head
over to https://hub.docker.com to create one.
Username: devnetstudent # This would be your username
Password:               # This would be your password
WARNING! Your password will be stored unencrypted in /home/ubuntu/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store
Login Succeeded
$
```

Save a Docker Image to a Registry

- Commit the running container with the `docker commit` command.

```
$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
PORTS                    NAMES
54c44606344c        sample-app-image    "/bin/sh -c 'python …"   4 seconds ago       Up 2 seconds
0.0.0.0:8080->8080/tcp    pythontest
$ docker commit pythontest sample-app
Sha256:bddc326383032598a1c1c2916ce5a944849d90e4db0a34b139eb315af266e68b
$
```

- Use the `docker tag` command to give the image the tag which was committed.

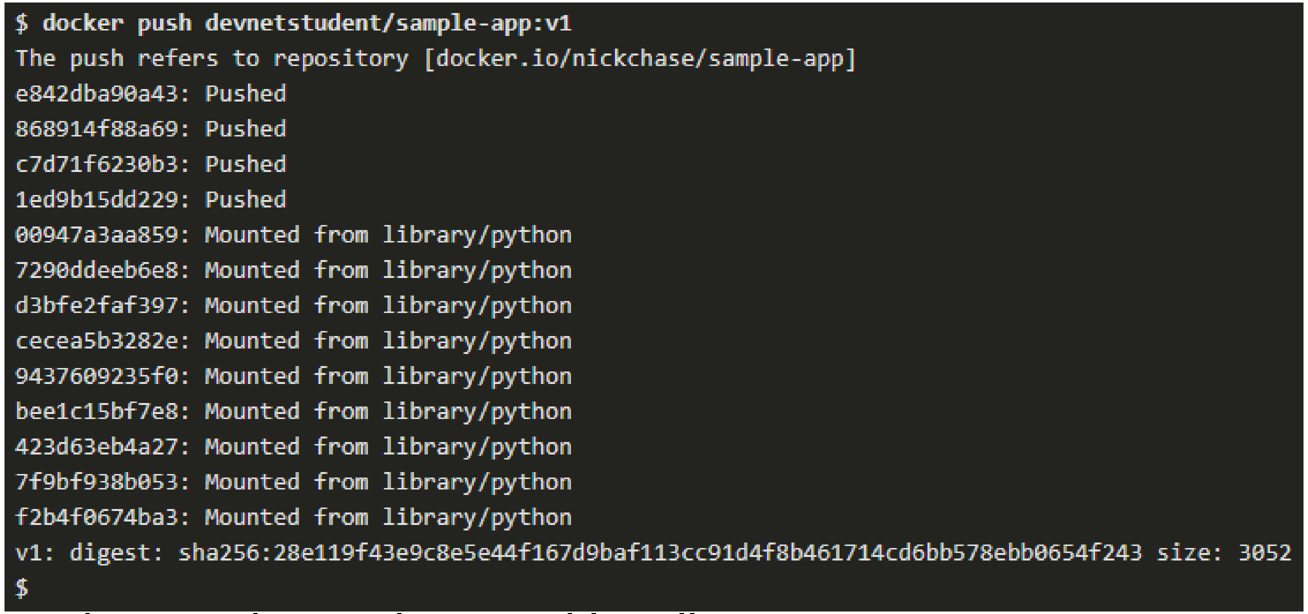
```
<repository>/<imagename>:<tag>
```

- The first part, the repository, is usually the username of the account storing the image. Next is the image name, and then finally the optional tag.

```
$ docker tag sample-app devnetstudent/sample-app:v1
$
```

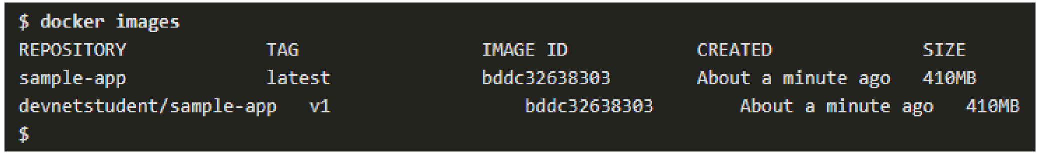
Save a Docker Image to a Registry

- Now the image is ready to be pushed to the repository.



```
$ docker push devnetstudent/sample-app:v1
The push refers to repository [docker.io/nickchase/sample-app]
e842dba90a43: Pushed
868914f88a69: Pushed
c7d71f6230b3: Pushed
1ed9b15dd229: Pushed
00947a3aa859: Mounted from library/python
7290ddeeb6e8: Mounted from library/python
d3bfe2faf397: Mounted from library/python
cecea5b3282e: Mounted from library/python
9437609235f0: Mounted from library/python
bee1c15bf7e8: Mounted from library/python
423d63eb4a27: Mounted from library/python
7f9bf938b053: Mounted from library/python
f2b4f0674ba3: Mounted from library/python
v1: digest: sha256:28e119f43e9c8e5e44f167d9baf113cc91d4f8b461714cd6bb578ebb0654f243 size: 3052
$
```

- Notice that the new image is stored locally.



```
$ docker images
REPOSITORY              TAG                 IMAGE ID            CREATED              SIZE
sample-app              latest              bddc32638303        About a minute ago   410MB
devnetstudent/sample-app   v1                   bddc32638303        About a minute ago   410MB
$
```

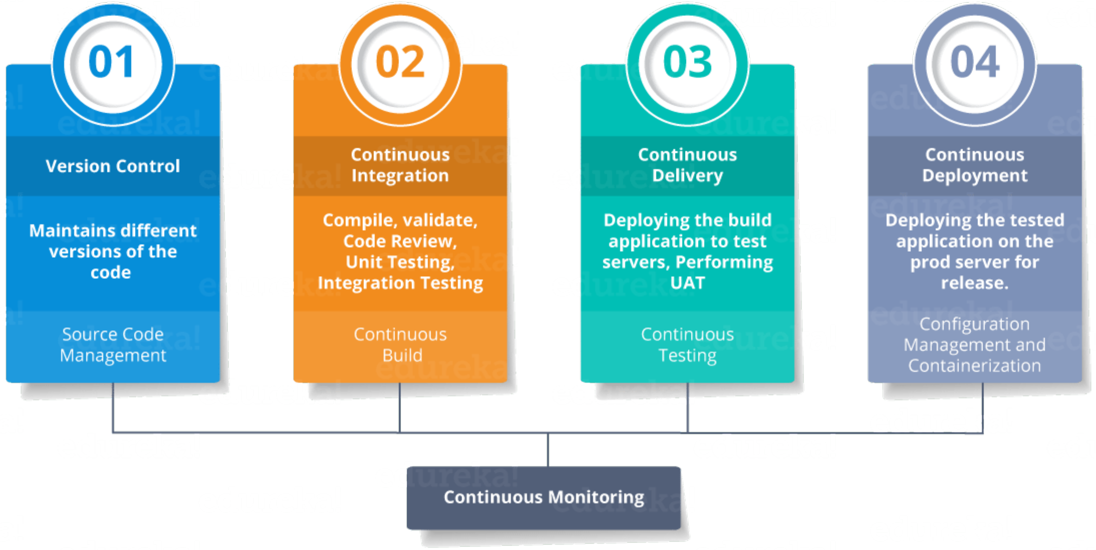
Create a Development Environment

- The development environment is meant to be convenient to the developer. It only needs to match the production environment where it is relevant.
- A development environment can consist of any number of tools from Integrated Development Environments (IDEs) to databases to object storage such as Eclipse to databases to object storage. The important part here is that it has to be comfortable for the developer.

6.3 Continuous Integration/Continuous Deployment (CI/CD)

Introduction to CI/CD

- **Continuous Integration/Continuous Deployment (CI/CD)** is a philosophy for software deployment that figures prominently in the field of DevOps.
- DevOps is about communication and making certain all members of the team are working together to ensure smooth operation.

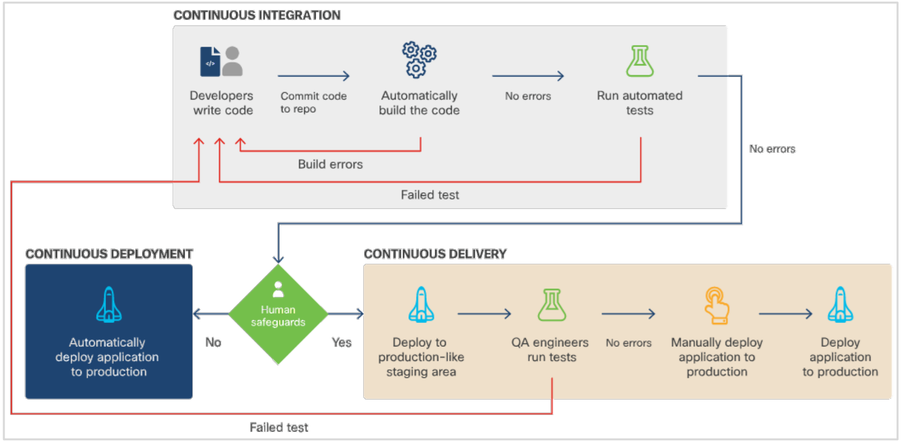

01
Version Control
Maintains different versions of the code
Source Code Management
02
Continuous Integration
Compile, validate, Code Review, Unit Testing, Integration Testing
Continuous Build
03
Continuous Delivery
Deploying the build application to test servers, Performing UAT
Continuous Testing
04
Continuous Deployment
Deploying the tested application on the prod server for release.
Configuration Management and Containerization
Continuous Monitoring

Introduction to CI/CD

- **Continuous Integration** enables the developers on the project to continually merge the changes with the main branch of the existing application.
- The Continuous Integration process provides a number of additional benefits:
  - Code compilation
  - Unit test execution
  - Static code analysis
  - Integration testing
  - Packaging and versioning
  - Publishing the version package to Docker Hub or other package repositories

Introduction to CI/CD

- **Continuous Delivery**
  - It is the process of developing in short sprints so that the code is always in a deployable state. It involves the following steps:
    - Step 1: Start with the version artifact created as part of the CI process.
    - Step 2: Automatically deploy the candidate version on staging.
    - Step 3: Run gating tests identified by the team or organization.
    - Step 4: If all gating tests pass, tag this build as suitable for production.



Introduction to CI/CD

- **Continuous Deployment**
  - Continuous Deployment is the ultimate expression of CI/CD.
  - It is a special type of Continuous Delivery in which, every version of software that is marked as ready for production gets deployed.
- **Preventing impact to users**
  - In order to avoid impacting users, or limit the impact, these deployment strategies can be used:
    - **Rolling upgrade**: The changes are periodically rolled out in such a way that they don't impact current users, and nobody should have to reinstall the software.
    - **Canary pipeline**: The new version is rolled out to a subset of users. If these users experience problems, the changes can be easily rolled back. If these users don't experience problems, the changes are rolled out to the rest of production.
    - **Blue-green deployment**: An entirely new environment (Blue) is created with the new code on it, but the old environment (Green) is held in reserve.

CI/CD Benefits

- The benefits of using CI/CD for development include:
  - Integration with agile methodologies
  - Shorter Mean Time To Resolution (MTTR)
  - Automated deployment
  - Less disruptive feature releases
  - Improved quality
  - Improved time to market

Example Build Job for Jenkins

- Deployment pipelines are normally created with a build tool such as Jenkins. These pipelines can handle tasks such as gathering and compiling source code, testing, and compiling artifacts such as tar files or other packages.
- Example build job for Jenkins
  - The fundamental unit of Jenkins is the project, also known as the job. Jobs are created to do all sorts of things, from retrieving code from a source code management repo to building an application using a script or build tool, to packaging it up and running it on a server.

Example Build Job for Jenkins

- To create a simple job that retrieves a version of the sample application from GitHub and runs the build script, perform the steps listed below:
  - Step 1: Create a New Item in the Jenkins interface by clicking the "create new jobs" link on the welcome page.
  - Step 2: Enter a name, choose Freestyle project (so that you have the most flexibility) and click OK.
  - Step 3: Scroll down to Source Code Management and select Git, then enter a GitHub repository URL for the Repository URL.
  - Step 4: Scroll down to Build and click Add Build Step. Choose Execute shell.
  - Step 5: In the Command box, add the command: `buildscript.sh`
  - Step 6: On the left-hand side, click Build Now to start the job.
  - Step 7: Move your mouse over the build number to get a pulldown menu that includes a link to the Console Output.

Example Build Job for Jenkins

- To create a second job that tests the build to ensure that it is working properly, perform the following steps:
  - Step 1: Click the Jenkins link and New Item to start a new job, then create another Freestyle job called TestAppJob.
  - Step 2: This time, leave the Source Code Management as None. But there is an option to set a Build Trigger so that this job runs right after the previous job, BuildAppJob.
  - Step 3: Scroll down and once again add a Build Step of Execute shell script.
  - Step 4: Add the following script as the command, using the IP address of an example Jenkins server and check to see if a condition is returned as true.

```
if [ "$(curl localhost:8000/test)" = "You are calling me from 172.17.0.1" ]; then
   exit 0
else
   exit 1
fi
```

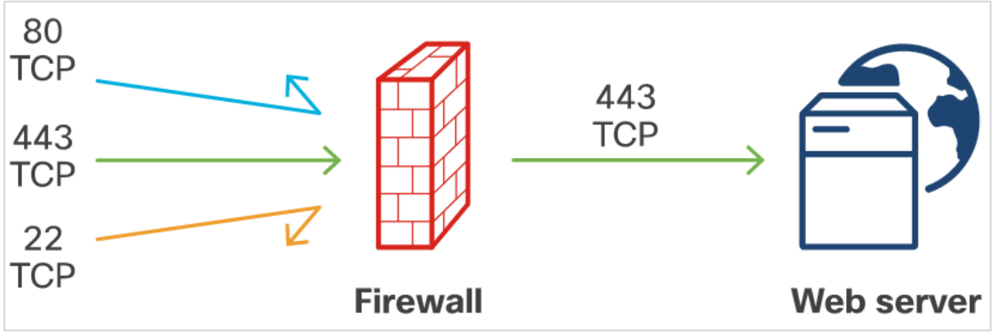
6.4 Networks for Application Development and Security

Introduction

- Networking accounts for all but the simplest of use cases such as cloud and container deployments.
- Some of the applications which needs to be considered for cloud deployment are given below:
  - Firewalls
  - Load balancers
  - DNS
  - Reverse proxies

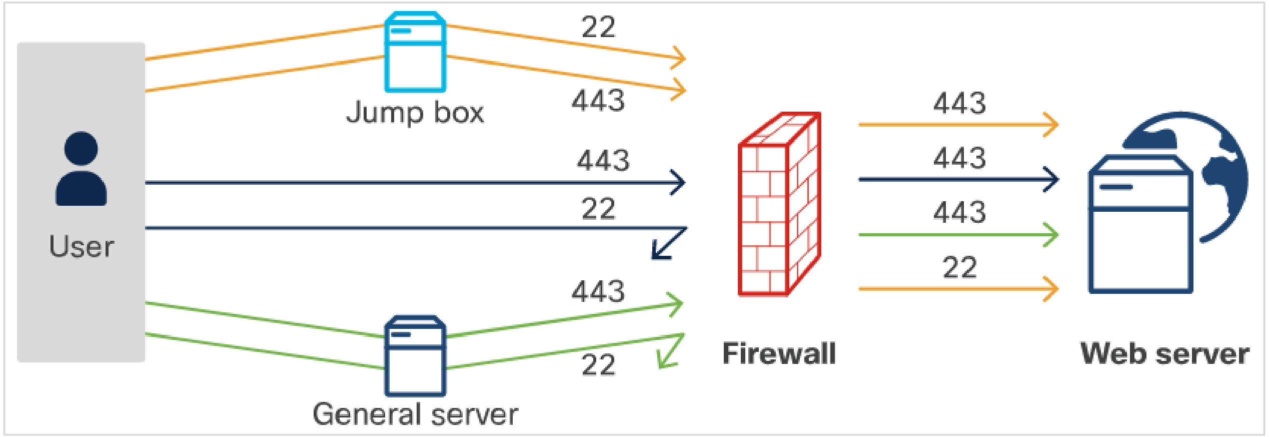
Firewall

- Firewalls are a computer's most basic defense against unauthorized access by individuals or applications. They can take any number of forms, from a dedicated hardware device to a setting within an individual computer's operating system.
- At its most basic level, a firewall accepts or rejects packets based on the IP addresses and ports to which they're addressed.
- Firewalls can be set up with specific "rules", which are layered on top of each other.
- A firewall can allow some connections and reject others.



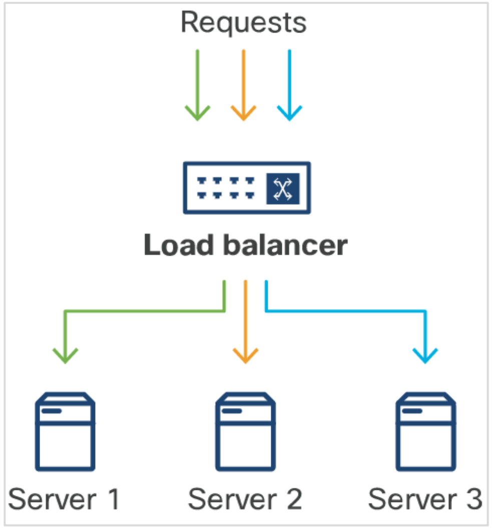
Firewall

- In some cases, you might set up your systems so that logins to sensitive systems can only come from a single machine. This is called a **jump box**.
- A jump box can be used to provide additional access while still providing an additional layer of security. It sets up the systems so that logins can only come from a single machine and everyone must log into that server first, then log into the target machine from there.



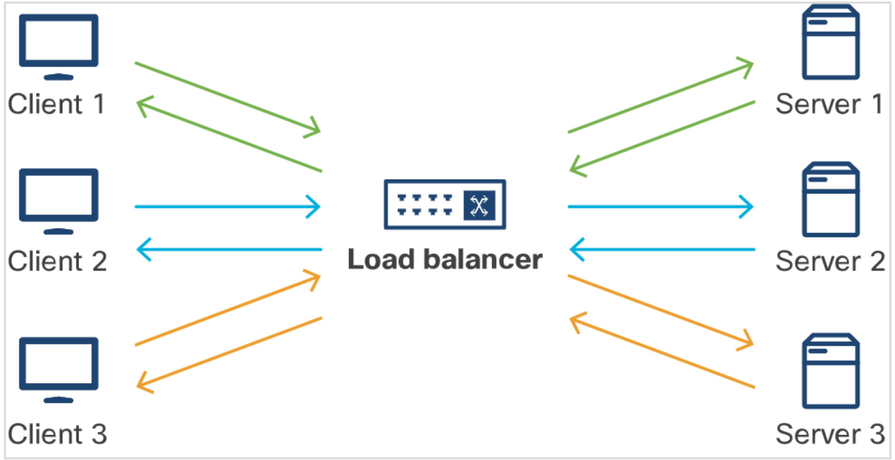
Load Balancer

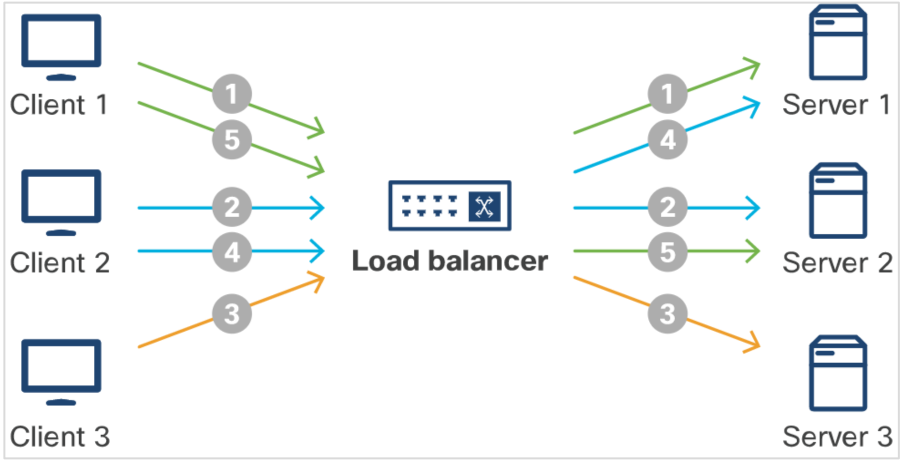
- A load balancer takes requests and balances them by spreading them out among multiple servers.
- A load balancer parcels out requests to different servers.
- Load balancers makes their decisions on which servers should get a particular request in a few different ways.



Load Balancer

- **Persistent sessions -** If an application requires a persistent session, a user needs to be logged in and the load balancer will send requests to the server handling the session.
- **Round robin -** With round robin load balancing, the server sends each request to the next server on the list.

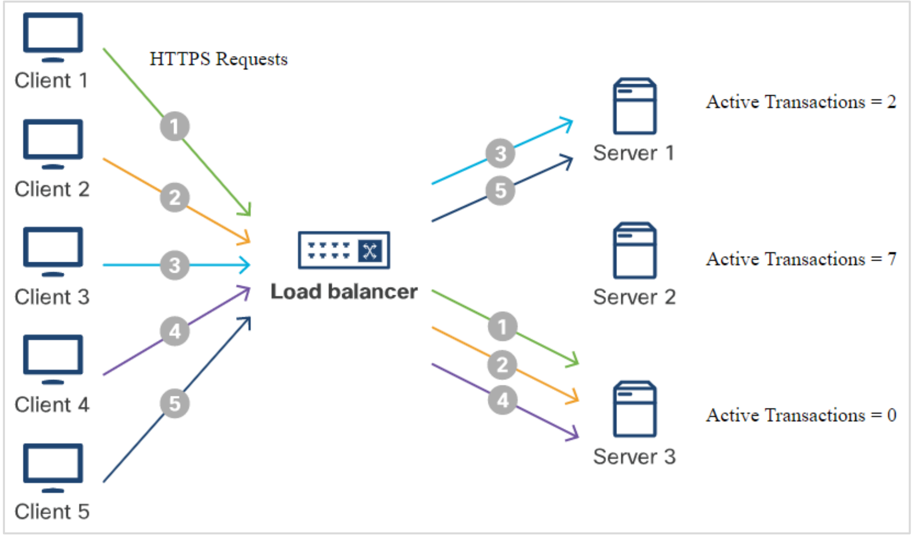


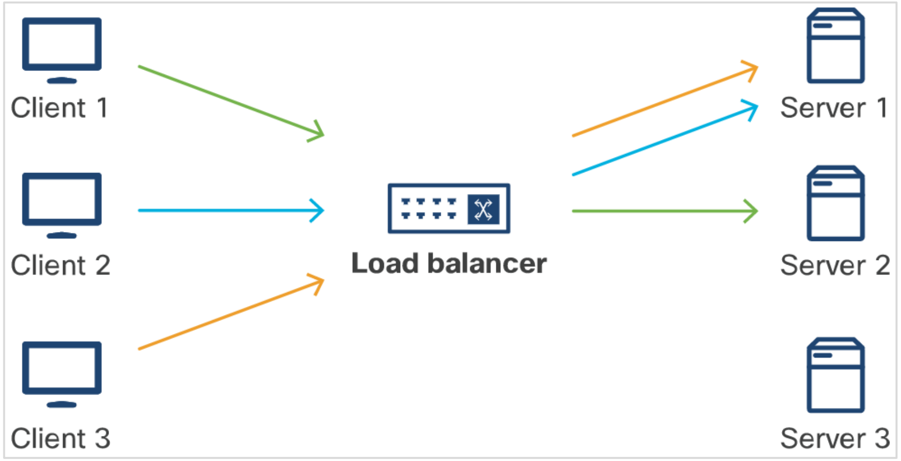


Load Balancer

Least connections - The load balancer sends request to the server that is the least busy - the least number of active connections.

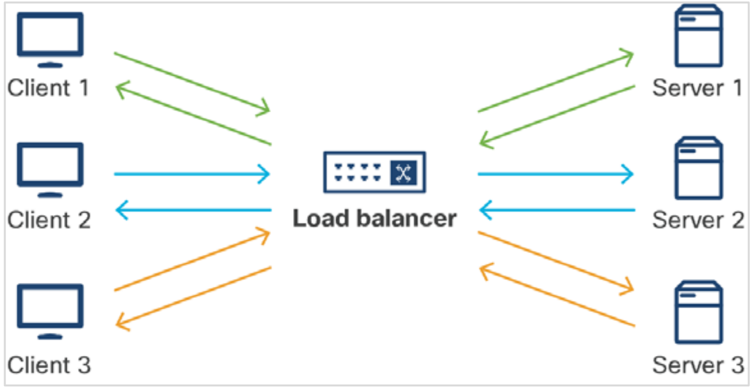
- **IP Hash -** With this algorithm, The load balancer makes a decision based on a hash (an encoded value based on the IP address of the request).



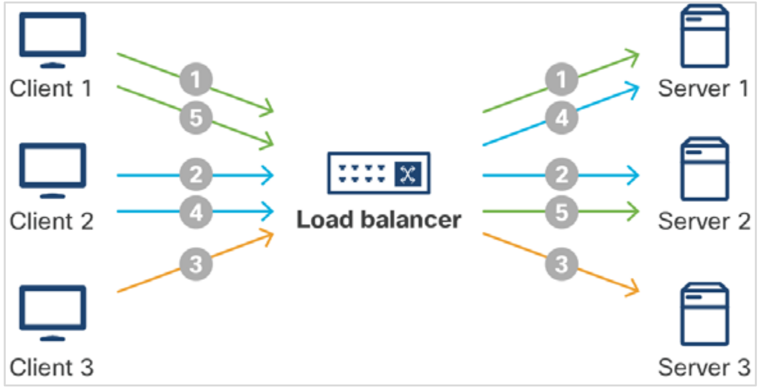


Load Balancer

- **Blue-green deployment -** Applies changes to a new production environment (blue) rather than making the changes on the existing production environment (green). A load balancer sends traffic to the blue environment when it is ready, and if issues arise, the load balancer can send traffic back to the green environment and changes can be rolled back.

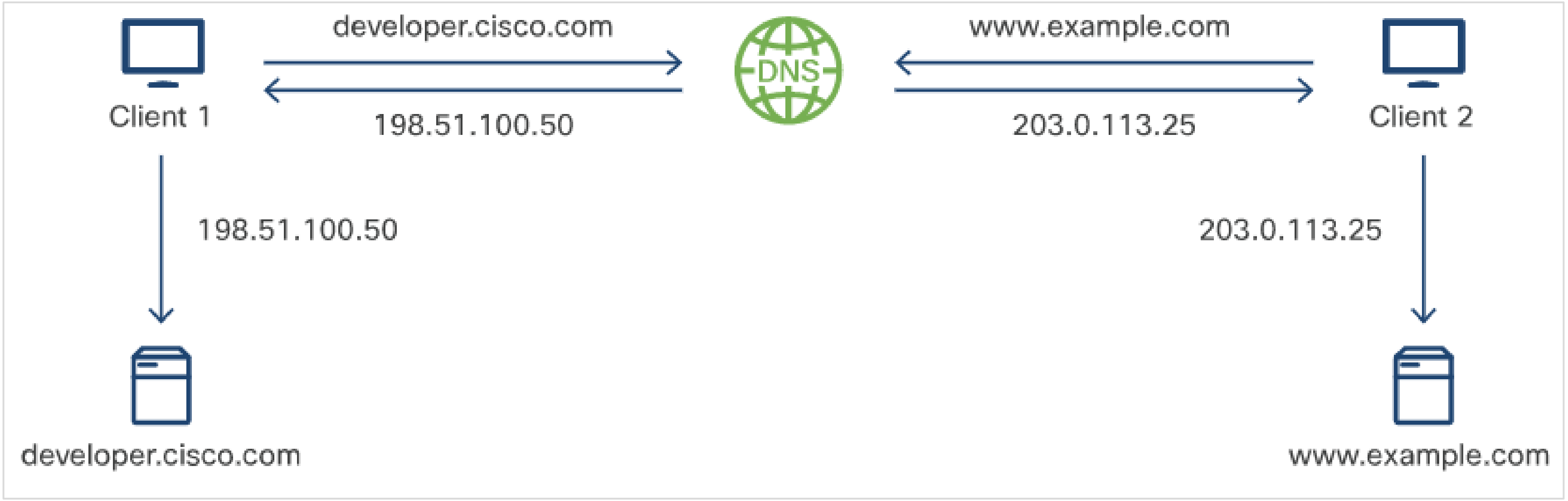


- **Canary deployment -** Diverts a small fraction of your traffic to the blue environment. A load balancer can then increase the amount of traffic diverted to the blue environment until issues are detected and traffic goes back to the old environment, or all servers and users are on the new environment, and the old one is retired or used for the next push.



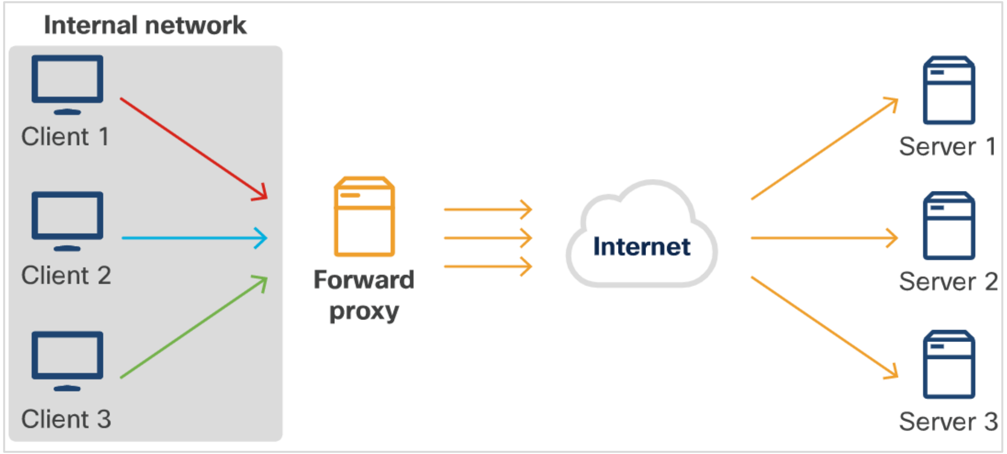
DNS

- Domain Name System (DNS) provides a way for the servers on the internet to translate human-readable names into machine-routable IP addresses. These IP addresses are necessary to actually navigate the internet.
- DNS translates hostnames into (made-up) IP addresses.



Reverse Proxy

- A reverse proxy is similar to a regular proxy, however, while a regular proxy works to make requests from multiple computers look like they all come from the same client, a reverse proxy works to make sure responses look like they all come from the same server.
- A reverse proxy can evaluate traffic and act accordingly. In this way, it is similar to, and can be used as, a firewall or a load balancer.



6.5 Securing Applications

Securing the Data

- **Best practices for storing encrypted data**
  - Data breaches occur when data is stored but not protected. When it comes to protecting data at rest, there are a few things to consider.
- **Encrypting data**
  - Data encryption ensures that when an unauthorized access is gained into the system, the data is not visible in its actual form. There are two methods for encrypting data:

One-way encryption	Two-way encryption
One-way encryption is simpler, in that you can easily create an encrypted value without necessarily using a specific key, but you cannot unencrypt it. You would use that for information you do not need to retrieve, just need to compare, such as passwords.	In Two-way encryption, you encrypt the data using a key, and then you can use that key (or a variation on it) to decrypt the data to get it back in plaintext. You would use this for information you would need to access in its original form, such as medical records or social security numbers.

Securing the Data

- **Software vulnerabilities**
  - Most developers are not experts in security and can accidentally code security vulnerabilities into the application. Ensure that someone in the organization is responsible for keeping up with the latest vulnerabilities and patching them as appropriate.
- **Storing too much data**
  - Unless the data is needed for an essential function, don't store it.
- **Storing data in the cloud**
  - Remember that when storing data in the cloud, it is stored in someone else's computer. Make sure that your cloud data is encrypted or otherwise protected.
- **Roaming devices**
  - Apps are increasingly on devices that even more portable than laptops, such as tablets and especially mobile phones. They are simply easier to lose. Be sure you are not leaving your data vulnerable by encrypting it whenever possible.

Securing the Data

- **Best practices for transporting data**
- Data is also vulnerable when it is being transmitted. The following can be used to prevent data vulnerability problems:
  - **SSH** - SSH provides authentication and encryption of messages between the source and target machines, making it difficult or impossible to snoop on the users' actions.
  - **TLS** - TLS provides message authentication and stronger ciphers than SSL.
  - **VPN** - A VPN keeps all application-related traffic inside the network, which acts as a proxy and encrypts all traffic to and from the user.

What is SQL Injection?

- SQL injection is a code injection technique that is used to attack data-driven applications, in which malicious SQL statements are inserted into an entry field for execution.
- SQL injection exploits a security vulnerability in an application's software. This attack allows attackers to spoof identity, tamper with existing data, allow the complete disclosure of all data on the system, destroy the data or make it otherwise unavailable, and become administrators of the database server.
- **SQL in Web Pages**
  - SQL injection is one of the most common web hacking techniques. It is the placement of malicious code in SQL statements, via web page input.
  - It occurs when a user is asked for input, like username/userid, and instead the user gives an SQL statement that is unknowingly executed on the database.

What is SQL Injection?

- This example creates a SELECT statement by adding a variable `uid` to a select string. The variable is fetched from user input using `request.args("uid")`.

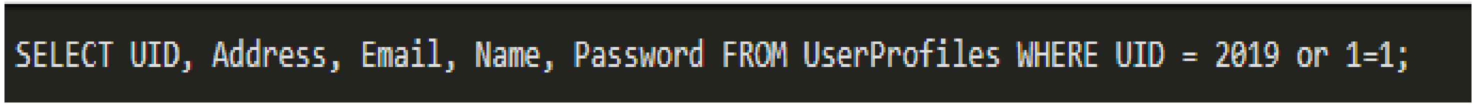
```
uid = request.args("uid");
str_sql = "SELECT * FROM Users WHERE UserId = " + uid;
```

- SQL Injection based on 1=1 is always true. Create an SQL statement to select user profile by UID, with a given UserProfile UID.
- If there is not input validator to prevent a user from entering "wrong" input, the user can enter some input as UID: `2019 OR 1=1`
- The output SQL statement will be:

```
SELECT * FROM UserProfiles WHERE UID = 2019 OR 1=1;
```

What is SQL Injection?

- The SQL statement above is valid, but will return all rows from the UserProfiles table, because OR 1=1 is always TRUE.
- If the UserProfiles table contains names, emails, addresses, and passwords, the SQL statement will be:

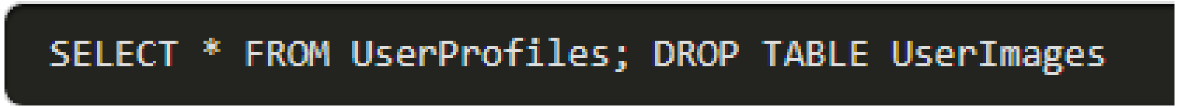


```
SELECT UID, Address, Email, Name, Password FROM UserProfiles WHERE UID = 2019 or 1=1;
```

- A malware creator or hacker might get access to all user profiles in database, by simply typing `2019 OR 1=1` into the input field.

What is SQL Injection?

- SQL Injection based on batched SQL statements
- Most databases support batched SQL statements. A batch of SQL statements is a group of two or more SQL statements, separated by semicolons.
- The SQL statement below will return all rows from the UserProfiles table, then delete the UserImages table.



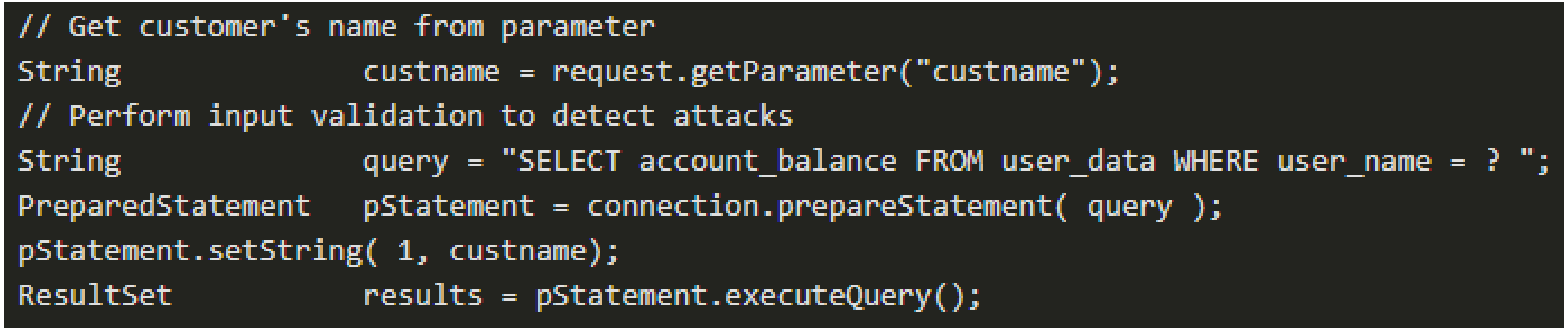
```
SELECT * FROM UserProfiles; DROP TABLE UserImages
```

How to Detect and Prevent SQL Injection

- SQL injection vulnerability exists because some developers do not care about data validation and security. There are tools that can help detect flaws and analyze code.
  - **Open source tools**: To detect a SQL injection attack easily, developers have created good detection engines such as SQLmap or SQLninja.
  - **Source code analysis tools**: Source code analysis tools, also known to as Static Application Security Testing (SAST) tools, are designed to analyze source code and/or compiled versions of code to help find security flaws such as buffer overflows, SQL Injection flaws, and others.
  - **Work with a database firewall**: Database firewalls detect SQL injections based on the number of invalid queries from a host, while there are OR and UNION blocks inside of request, or others.

How to Detect and Prevent SQL Injection

- **Use prepared statements**
  - Prepared statements with variable binding - also known as parameterized queries - are used by developers for writing database queries. Parameterized queries force the developer to first define all the SQL code, and then pass parameter to the query.
  - Prepared statements ensure that an attacker is not able to change the intent of a query, even if the SQL commands are inserted by an attacker.



```
// Get customer's name from parameter
String              custname = request.getParameter("custname");
// Perform input validation to detect attacks
String              query = "SELECT account_balance FROM user_data WHERE user_name = ? ";
PreparedStatement   pStatement = connection.prepareStatement( query );
pStatement.setString( 1, custname);
ResultSet           results = pStatement.executeQuery();
```

How to Detect and Prevent SQL Injection

- **Use Stored Procedures**
  - Stored procedures are not always safe from SQL injection.
  - The difference between prepared statements and stored procedures is that the SQL code for a stored procedure is defined and stored in the database itself, and then called from the application.
  - Both of these techniques have the same effectiveness in preventing SQL injection.
- **Whitelist Input Validation**
  - Various parts of SQL queries are not legal locations for the use of bind variables.
  - In such situations, input validation or query redesign is the most appropriate defense.
  - If user parameter values are used for targeting different table names and column names, then the parameter values should be mapped to the legal/expected table or column names to ensure unvalidated user input does not end up in the query.

How to Detect and Prevent SQL Injection

- **Escaping all user-supplied input**
  - This technique should only be used as a last resort when none of the techniques are feasible and involves escaping user input before putting it in a query.
  - The Escaping works in such a way that each DBMS supports one or more character escaping schemes specific to certain kinds of queries.
  - There are libraries and tools used for Input Escaping.
  - The ESAPI libraries make it easier for programmers to retrofit security into applications and serve as a solid foundation.

How to Detect and Prevent SQL Injection

- **Additional defenses**
  - To provide defense in depth, these additional defenses can be adopted:
    - **Least privilege**: The privileges assigned to every database account should be minimized in order to reduce the potential damage of a successful SQL injection attack. Minimizing the privileges will reduce the unauthorized access attempts, even when an attacker is not trying to use SQL injection as part of their exploit.
    - **Multiple database users**: Web applications designers should avoid using the same owner/admin account in the web applications to connect to the database. Different DB users could be used for different web applications.
    - **SQL views**: SQL views is used to further increase the access detail by limiting read access to specific fields of a table or joins of tables.

Secure the Application

- The Open Web Application Security Project (OWASP) is focused on providing education, tools, and other resources to help developers avoid security problems in web-based applications. Resources provided by OWASP include:

Tools	
OWASP Zed Attack Proxy (ZAP)	Looks for vulnerabilities during development
Dependency Check	Looks for known vulnerabilities in your code
OWASP DefectDojo	Streamlines the testing process
Code Projects	
OWASP ModSecurity Core Rule Set (CRS)	Generic attack detection rules that can be used with web application firewalls
OWASP CSRFGuard	Helps prevent Cross-Site Request Forgery (CSRF) attacks
Documentation Projects	
OWASP Application Security Verification Standard	Provides a basis for testing web application technical security controls
OWASP Top Ten	Describes the 10 most common security issues in web applications
OWASP Cheat Sheet Series	Explains how to mitigate command security issues in web applications

Cross-Site Scripting (XSS)

- Cross site scripting attacks happen when user-submitted content that has not been sanitized is displayed to other users.
- The most obvious version of this exploit is where one user submits a comment that includes a script that performs a malicious action and anyone who views the comments page has that script executed on their machine.
- Nowadays, the bigger problem is that the users are dealing with more than the data that is stored in the database, or “Stored XSS Attacks.” For example, consider this page, which displays content from a request parameter:

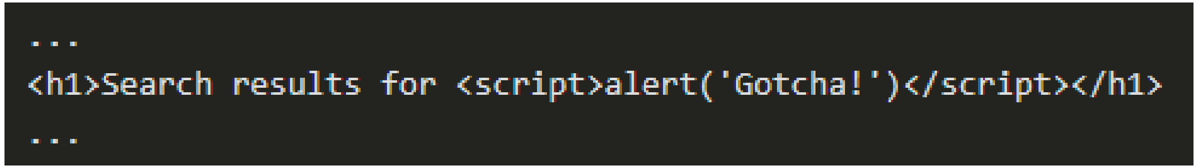
```
...
<h1>Search results for {{ request.args[‘search_term’] }}</h1>
{ for item in cursor }
…
```

- A hacker could trick someone into visiting the page with a link in an email that provides malicious code in a parameter:

```
http://www.example.com?search_term=%3Cscript%3Ealert%28%27Gotcha%21%27%29%3C%2Fscript%3E
```

Cross-Site Scripting (XSS)

- This link, which includes a url encoded version of the script, would result in an unsuspecting user seeing a page of:



```
...
<h1>Search results for <script>alert('Gotcha!')</script></h1>
...
```

- This is called a **Reflected XSS Attack**. To prevent a reflected XSS attack, the main strategy is to sanitize content where possible, and if it cannot be sanitized, do not display it.
- OWASP recommends never displaying untrusted content in the following locations:
  - Inside script tags
  - Inside comments
  - As part of attribute names
  - As part of tag names
  - In CSS (within style tags)

Cross-Site Scripting (XSS)

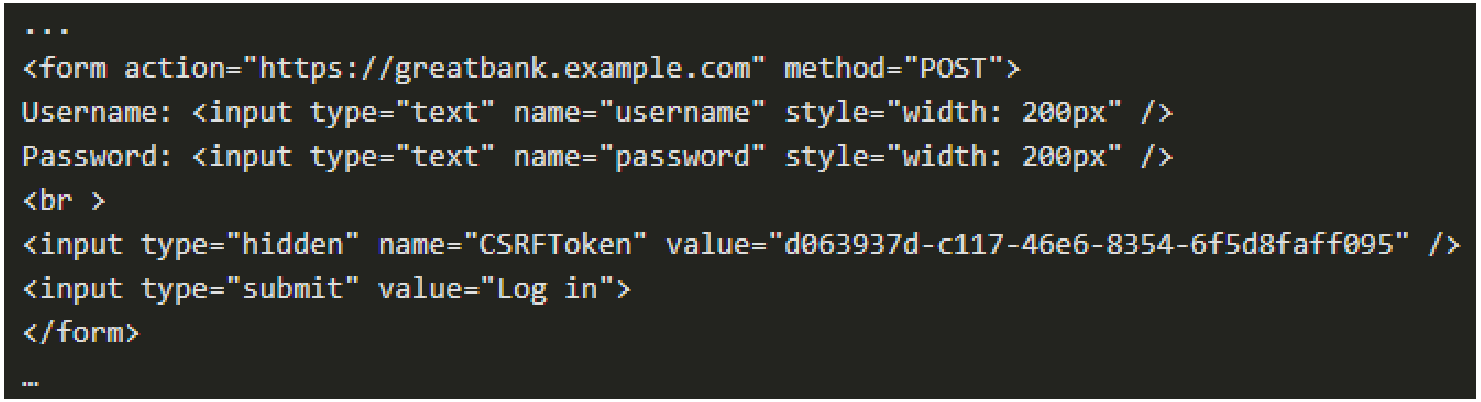
- The content can be displayed in some locations, if it is sanitized first. These locations include:
  - Content of an HTML tag
  - Value of an attribute
  - Variable within Javascript
- Sanitizing content can be a complicated process to get right, as there are a wide variety of options an attacker has.

Cross-Site Request Forgery (CSRF)

- Another type of attack that shares some aspects of XSS attacks is Cross Site Request Forgery (CSRF), sometimes pronounced “Sea Surf.”
- In both cases, the attacker intends for the user to execute the attacker’s code, usually without even knowing it.
- The main difference is that CSRF attacks are typically aimed not at the target site, but rather at a different site, one into which the user has already authenticated.
- An interesting aspect of CSRF is that the attacker never actually gets the results of the attack. They can only judge the results after the fact, and they have to be able to predict what the effects will be to take advantage of a successful attack.

Cross-Site Request Forgery (CSRF)

- One method to prevent CSRF attacks is to include a hidden token that must accompany any requests from the user.



```
...
<form action="https://greatbank.example.com" method="POST">
Username: <input type="text" name="username" style="width: 200px" />
Password: <input type="text" name="password" style="width: 200px" />
<br >
<input type="hidden" name="CSRFToken" value="d063937d-c117-46e6-8354-6f5d8faff095" />
<input type="submit" value="Log in">
</form>
…
```

- That `CSRFToken` has to accompany every request from the user for it to be considered legitimate as it is impossible for the attacker to predict that token.

The OWASP Top Ten

- OWASP list of attacks include:
  - **Injection**: This includes all sorts of injection attacks that can be prevented by using parameterized APIs, escaping user input, and by using LIMIT clauses.
  - **Broken Authentication**: This relates to multiple problems with user credentials. These attacks can be prevented by avoiding default passwords, using multi-factor authentication, and by using techniques like lengthening waiting periods after failed logins.
  - **Sensitive Data Exposure**: This refers to scenarios when attackers steal sensitive information. Such scenarios can be prevented by storing as little personal information as possible, and by using encryption.
  - **XML External Entities (XXE)**: These are attacks made possible an XML feature that permits incorporating external information using entities, and can be prevented by disabling XML Entity and DTD processing, or by using JSON format.

The OWASP Top Ten

- **Broken Access Control**: This refers to the need to ensure that an application that enables users to circumvent existing authentication requirements should not be built and can be avoided by protecting all resources and functions on the server side.
- **Security Misconfiguration**: This refers to the need to ensure that the system itself is properly configured. Prevention of these types of problems requires careful, consistent hardening of systems and applications.
- **Cross-Site Scripting (XSS)**: This refers to the ability for an attacker to use the dynamic functions of a site to inject malicious content into the page. These attacks can be prevented by carefully considering where to include the untrusted content as well as sanitizing any untrusted content.
- **Insecure Deserialization**: This describes issues that can occur if attackers can access, and potentially change, serialized versions of data and objects. To prevent such issues, do not accept serialized objects from untrusted sources.

The OWASP Top Ten

- **Using Components with Known Vulnerabilities**: Most of the core functions are probably been written and included in an existing software package, and it is probably open source. Many of the packages that are available also include publicly available exploits. To fix this, ensure that they are using only necessary features and secure packages.
- **Insufficient Logging and Monitoring**: It is important to ensure that the logs are in a common format so that they can be easily consumed by reporting tools, and that they are auditable to detect tampering.

Evolution of Password Systems

- **Simple Plaintext Passwords**
  - The first passwords were simple plaintext passwords that allowed multiple users using the same core processor to have unique privacy settings.
  - Plaintext is an insecure way of storing passwords. If the database was hacked, the user's passwords would be exposed to hackers directly.
- **Password Hashing**
  - Storing passwords is risky and complex at the same time.
  - A simple approach to storing passwords is to create a table in the database that maps a username with a password.
  - The security strength and resilience of this model depends on password storage format which is cleartext.
  - Storing passwords in cleartext is the equivalent of writing them down in a piece of digital paper. If an attacker breaks into the database and steal the passwords table, the attacker could then access each user account.

Evolution of Password Systems

Hashing

- Hashing is a more secure way to store a password in which it is transformed into data that cannot be converted back to the original password.
- As stated by OWASP, hash functions used in cryptography have the following key properties:
  - It is easy and practical to compute the hash, but difficult or impossible to re-generate the original input if only the hash value is known.
  - It's difficult to create an initial input that would match a specific desired output.

Salted password

- To guarantee the uniqueness of the passwords, increase their complexity, a salt, which is simply random data, is added to the input of a hash function.

Evolution of Password Systems

Using cryptographic hashing for more secure password storage

- A critical property that makes hash functions suitable for password storage is that they are deterministic.
- A deterministic function is a function that, given the same input, always produces the same output. This is vital for authentication because one needs to have the guarantee that a given password will always produce the same hash. Otherwise, it would be impossible to consistently verify user credentials with this technique.

Adding salt to password hashing

- A salt is added to the hashing process to force hash uniqueness thereby increasing complexity without increasing user requirements, and mitigating password attacks such as rainbow tables.
- The unique hash produced by adding the salt can protect against different attack vectors, while slowing down dictionary and brute-force attacks.

Evolution of Password Systems

- **Mitigating password attacks with a salt**
  - To mitigate the damage that a rainbow table or a dictionary attack could do, salt the passwords.
  - According to OWASP Guidelines, a salt is a fixed-length cryptographically-strong random value that is added to the input of hash functions to create unique hashes for every input, regardless of whether the input is unique.
  - Let’s say that you have password `devnet_password1` and the salt `salt706173776f726473616c74a`
  - You can salt that password by either appending or prepending the salt to it.

Evolution of Password Systems

- Additional factors for authentication
- Incorporating other authentication factors confuses the hackers who may have cracked the password. Some of these factors are as follows:
  - **Single-factor authentication (SFA)**
    - Single-factor authentication is the simplest form of authentication methods, using which, a person matches one credential to verify himself or herself online. The most popular example of this would be a password (credential) to a username.
    - SFA has its risks as Online sites can have users' passwords leaked by a hacker. A malicious user may guess the password as they know the user personally, or as they were able to find out certain things about the user.
    - A malicious user may also crack the password by using a bot to generate the right combination of letters/numbers to match the users' simple, secret identification method.

Evolution of Password Systems

- **Two-factor authentication (2FA)**
  - Two-factor authentication uses the same password/username combination, but with the addition of being asked to verify the identity of the persons by using something owned by them only such as a mobile device.
- **Multi-factor authentication (MFA)**
  - Multi-factor authentication (MFA) is a method of computer access control in which a user is only granted access after successfully presenting several separate pieces of evidence to an authentication mechanism.
  - At least two of the mentioned categories are required for MFA: knowledge; possession, and inherence.
  - 2FA is just a type of MFA where you only need two pieces of evidence, two “factors”.

Password Cracking

- The techniques for finding a password that allows entry is known as cracking the security intended by the password. The following are some of the techniques:
  - **Password guessing**
    - Password guessing is an online technique that involves attempting to authenticate a particular user to the system.
    - It may be detected by monitoring the failed login system logs.
    - Account lockouts are used to prevent an attacker from being able to simply guess the correct password by attempting a large number of potential passwords.
  - **Dictionary attack**
    - A dictionary attack is based on trying all the strings in a pre-arranged listing, derived from a list of words such as in a dictionary.
    - These succeed because many people have a tendency to choose short passwords that are ordinary words or common passwords.

Password Cracking

- **Pre-computed dictionary attack or rainbow table attack**
  - It is possible to achieve a time/space tradeoff by pre-computing a list of hashes of dictionary words and storing these in a database using the hash as the key.
  - Pre-computed dictionary attacks are effective when a large number of passwords are to be cracked.
  - Pre-computed dictionary attacks can be thwarted by the use of salt, a technique that forces the hash dictionary to be recomputed for each password sought, making pre-computation infeasible, provided the number of possible salt values is large enough.
- **Social engineering**
  - Social engineering for password cracking involves a person convincing or tricking another person for providing access to the attacker.

Password Cracking

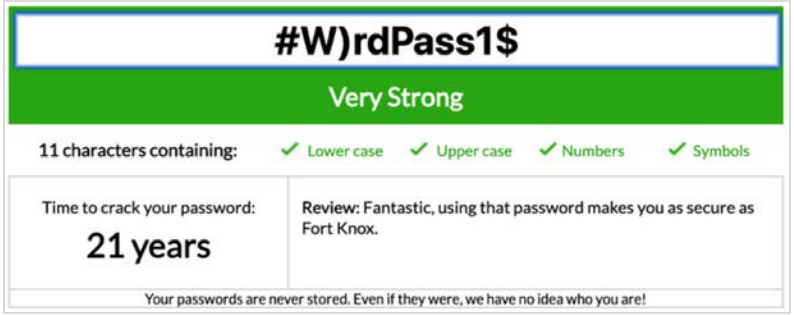
- Six key principles of human influence
  - **Reciprocity** – Our social norms mean that we tend to return a favor when asked.
  - **Commitment and consistency** – When people commit, whether in person, in writing, or on a web site, they are more likely to honor that commitment in order to preserve their self-image.
  - **Social proof** – When people see someone else doing something, such as looking up, others will stop to do the same.
  - **Authority** – This authority principle means that attackers who seem to be authoritative or representing an authority figure are more likely to gain access.
  - **Liking** – Likable people are able to persuade others more effectively. People are easily persuaded by familiar people whom they like.
  - **Scarcity** – When people believe that something is limited in amount, people will act positively and quickly to pick up the desired item.

Password Cracking

- There are four social engineering vectors, or lines of attack, that can take advantage of these influence principles.
  - **Phishing** means the person is fraudulently gaining information, especially through requests for financial information. Often the attempts look like a real web site or email, but link to a collector site instead.
  - **Vishing** stands for voice phishing, so it is associated with voice phone calls to gather private personal information for financial gain.
  - **Smishing** involves using SMS text messaging for both urgency and asking for a specific course of action, such as clicking a fake link or sending account information.
  - **Impersonation** involves in-person scenarios such as wearing a service provider uniform to gain inside access to a building or system.

Password Cracking

- **Password strength** - Password strength is the measure of a password’s efficiency to resist password cracking attacks. The strength of a password is determined by:
  - **Length**: This is the number of characters the password contains.
  - **Complexity**: This means it uses a combination of letters, numbers, and symbols.
  - **Unpredictability**: Something that can be guessed easily by an attacker.
- Here, the password #W)rdPass1 has strength and it would take about 21 years to crack it.



#W)rdPass1$

Very Strong

11 characters containing: ✓ Lower case ✓ Upper case ✓ Numbers ✓ Symbols

Time to crack your password:
21 years

Review: Fantastic, using that password makes you as secure as Fort Knox.

Your passwords are never stored. Even if they were, we have no idea who you are!

Password Cracking

- **Password strength checkers and validation tools**
  - The password strength validation tool is built in with password system to make sure the user's password is compatible with latest identity management guidelines.
  - Password manager is the tool to ensure the strength of the password.
- **Best practices**
  - There are a few best practices to secure user login attempts. It includes notifying users of suspicious behavior, limiting the number of password and username login attempts.

Password Cracking

- NIST Digital Identity Guidelines
- Here's a brief summary of the NIST 800-63B Digital Identity Guidelines:
  - 8-character minimum when a human sets it and 6-character minimum when set by system/service.
  - Support at least 64 characters maximum length and all ASCII characters.
  - Truncation of the password shall not be performed when processed.
  - Check chosen password with known password dictionaries.
  - Allow at least 10 password attempts before lockout.
  - No complexity requirements, password expiration period, password hints.
  - No SMS for two-factor authentication, knowledge-based authentication.

6.6 Summary: Application Deployment and Security

What Did I Learn in this Module?

- **Understanding Deployment Choices with Different Models**
  - Large organizations use a four-tier structure: development, testing, staging, and production.
  - The options to deploy the software are bare metal, virtual machines, containers, and serverless computing.
  - On-premises means any system that's within the confines of your building.
  - Clouds provide self-service access to computing resources, such as VMs, containers, and even bare metal.
  - The advantage of a private cloud is that the user has complete control over where it is located.
  - A public cloud is essentially the same as a private cloud, but it is managed by a public cloud provider.
  - Hybrid cloud is used to bridge a private cloud and a public cloud within a single application.
  - An edge cloud moves computing closer to where it's needed.

What Did I Learn in this Module?

- **Creating and Deploying a Sample Application**
  - A container is way of encapsulating everything needed to run the application, so that it can easily be deployed in a variety of environments.
  - Docker is a way of creating and running that container.
  - The development environment is meant to be convenient to the developer; it only needs to match the production environment
  - A development environment can consist of any number of tools, from IDEs to databases to object storage.

What Did I Learn in this Module?

- **Continuous Integration/Continuous Deployment (CI/CD)**
  - CI/CD is a philosophy for software deployment that figures prominently in the field of DevOps.
  - Continuous Integration all the developers on the project, continually merge your changes with the main branch of the existing application.
  - A deployment pipeline, can be created with a build tool such as Jenkins.
- **Networks for Application Development and Security**
  - The applications you need to consider when it comes to cloud deployment include: Firewalls, Load balancers, DNS, and Reverse proxies.
  - At its most basic level, a firewall accepts or rejects packets based on the IP addresses and ports to which they're addressed.

What Did I Learn in this Module?

- **Securing Applications**
  - Securing data in two methods by encrypting data: one-way encryption, and two-way encryption.
  - SQL injection must exploit a security vulnerability in an application's software.
  - A more secure way to store a password is to transform it into data that cannot be converted back to the original password, known as hashing.
  - By cryptography password are made to be secure.

Elk Technologies

https://elktech.org

SCOTT STEPHENSON