ECE 6132

Secure Cloud Computing

Security concerns and best practices for cloud computing and cloud services; cloud computing architectures, risk issues and legal topics; data security; internal and external clouds; information security frameworks and operations guidelines.

Dr. M

Welcome to SEAS Online at George Washington University

ECE-6132 class will begin shortly

- **Audio:** To eliminate background noise, please be sure your audio is muted. To speak, please click the hand icon at the bottom of your screen (**Raise Hand**). When instructor calls on you, click microphone icon to unmute. When you've finished speaking, ***be sure to mute yourself again***.
- **Chat:** Please type your questions in Chat.
- **Recordings:** As part of the educational support for students, we provide downloadable recordings of each class session to be used exclusively by registered students in that particular class for their own private use. Releasing these recordings is strictly prohibited.

Class-3

Getting ready for the course

Agenda

While the first week covers the traditional cloud computing architecture, the second week focuses on modern cloud computing services such as containers, Kubernetes, and serverless functions. We will create these services in a commercial cloud environment using a Python application capable of ingesting Cryptocurrency data for generating buy and sell signals. We will wrap the lecture by identifying the security vulnerabilities in manually created containers and serverless environments.

Topics include:

- Operating system
- Namespace and control groups
- Container management
- Secure vs. real time OS
- Unikernels
- Serverless functions
- Container security roles

Prerequisites

Software Install

- VirtualBox - https://www.virtualbox.org/wiki/Downloads
- Vagrant - https://www.vagrantup.com/downloads
- Docker - https://docs.docker.com/get-docker/
- Kubernetes - https://kubernetes.io/docs/tasks/tools/install-kubectl-macos/
- Minikube - https://minikube.sigs.k8s.io/docs/start/
- Java - https://www.oracle.com/java/technologies/downloads
- Python - https://www.python.org/downloads/

Secure Computing in the Cloud

Challenges

1. **Differences in dev and production environment is exaggerated with the cloud.**
2. Security is a shared responsibility in the cloud.
3. Sizing resources for application and managing cost efficiencies is difficult.

Secure Computing in the Cloud

“Hello World”

- Java: (`master-class/building-blocks/java/HelloWorld.java`)

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World");
    }
}
```

- Python: (`master-class/building-blocks/python/HelloWorld.py`)

```
print(“Hello World”)
```

Secure Computing in the Cloud

COMPILE, PACKAGE, AND VERIFY

- Java: JAR (`master-class/building-blocks/java/Makefile`)

```
javac HelloWorld.java
jar cvf HelloWorld.jar HelloWorld.class
java -cp HelloWorld.jar HelloWorld
```

- Python: PIP (`master-class/building-blocks/python/Makefile`)

```
python setup.py sdist
pip install dist/HelloWorld-0.1.tar.gz
python HelloWorld.py
```

Secure Computing in the Cloud

WHAT DO WE NEED TO TEST OUR APPLICATION?

- *Code*: Java jar and Python package
- *Runtime*: “jre” and “python”
- *Dependencies*: Libraries with specific version
- *Configure*: Config files, start up scripts, and log directories
- *Monitoring*: service monitoring and auto-restarting
- *Servers*: instances similar to production

Secure Computing in the Cloud

IS THIS A SURPRISE? NO

- *Code*: Artifactory, Maven, Pip repo
- *Runtime*: Chef, Ansible, Cloud-init, Bash scripts
- *Dependencies*:
    - System dependencies: Chef, Ansible
    - Application dependencies: Big jar, pip wheel
- *Configure*: Native SystemD, Chef, Ansible
- *Monitoring*: Vagrant VMWare/VirtualBox and Consul
- *Servers*: Vagrant with VMWare/VirtualBox
-

Secure Computing in the Cloud

SO, WHAT IS THE PROBLEM?

1. None of these solutions can replicate the production environment on the user's development laptop because they are all distinct solutions.

2. It takes too much time to learn and master different technologies. It is hard to keep track of supporting technologies if you're a software developer.

Secure Computing in the Cloud

WHEN THINGS DON'T WORK?

The Dev says:

"it works on my laptop,
must be a production issue"

The Ops says:

"code is not production ready yet"

The synthesis:

"we failed"

Secure Computing in the Cloud

SOLUTION? WE NEED A NEW PACKAGING TECHNOLOGY

- Packaging technology that has potential to integrate:
  - Code
  - Run time environment
  - Both system and application dependencies
  - Configure once and run it any where support
  - Built in health and healing support
  - Ability to spin servers on laptop

Secure Computing in the Cloud

WHAT IS THE SOLUTION?

Problem: Differences in dev and production environment is exaggerated with the cloud.

Solution: App Integrated Images

Secure Computing in the Cloud

Challenges

1. Differences in dev and production environment is exaggerated with the cloud.
2. **Security is a shared responsibility in the cloud.**
3. Sizing resources for application and managing cost efficiencies is difficult.

Secure Computing in the Cloud

WHY DO WE NEED ISOLATION?

Every process has access to all the resources from Global namespace by default.

```
$ cd master-class/building-blocks/namespace
$ vagrant up
$ vagrant ssh
$ /vagrant/process-list.py
```

(You may see 50+ processes)

- Do you own those processes? Nope!
- Should you have access to such information during run time?

Secure Computing in the Cloud

Challenges

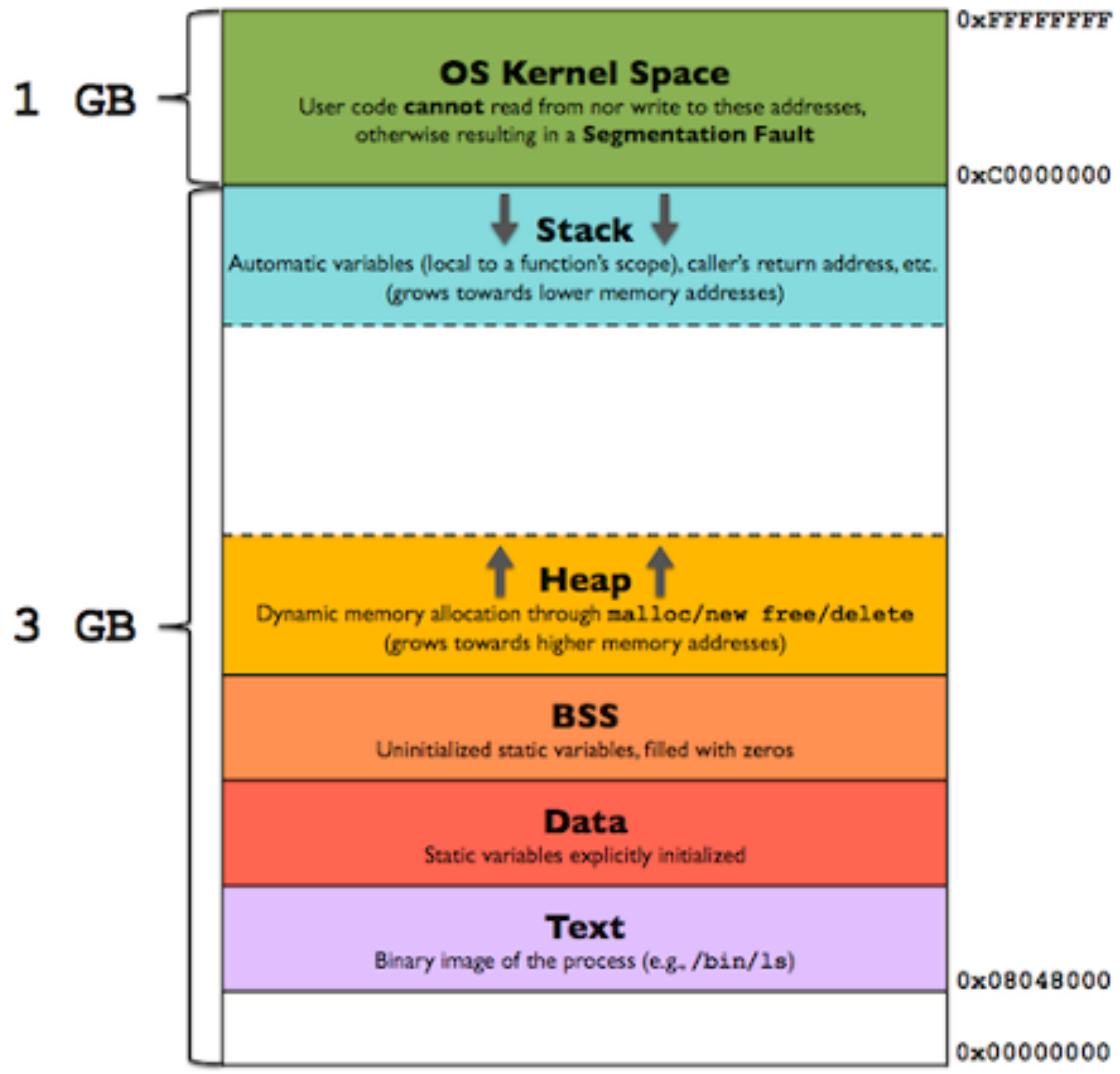
- Restrict a process to a view only that is relevant resources on the system.

Secure Computing in the Cloud

WHY DO WE NEED ISOLATION?

- It used to be a big problem.
- Solutions:
  - Virtual Address Space
  - Address Space Layout Randomization
- How about Storage?
- How about Network?

  … the list goes on!



(Courtesy: http://logicmoment.com/memory-map-of-c-program)

Secure Computing in the Cloud

HOW DO WE SOLVE IT TODAY?

- We avoid the problem by using a single server or VM per application, but at a huge financial cost.
- Based on the monitoring statistics, ***our resource utilization is less than 20%.***

Secure Computing in the Cloud

HOW DO WE SOLVE IT TODAY?

Problem: Security is a shared responsibility in the cloud.

Solution: Namespaces

Secure Computing in the Cloud

HOW DO WE SOLVE IT TODAY?

```
$ cd master-class/building-blocks/namespace
$ vagrant up
$ vagrant ssh
$ /vagrant/process-list.py
```

(You may see 50+ processes)

```
$ sudo /vagrant/isolate.sh /vagrant/process-list.py
```

(You should see 2 processes)

Secure Computing in the Cloud

Challenges

1. Differences in dev and production environment is exaggerated with the cloud.
2. Security is a shared responsibility in the cloud.
3. **Sizing resources for application and managing cost efficiencies is difficult.**

Secure Computing in the Cloud

WHAT IS THE PROBLEM?

- Laptops are not as resource heavy as servers.
- We need to avoid a situation where one service is consuming all the available resources such as CPU, Memory, IO and Network.
- Restricting access using namespaces controls:
    - What a process can see
    - NOT, how much it can utilize
- This problem is relevant when resources are shared.

Secure Computing in the Cloud

WHAT IS THE SOLUTION?

Control Groups

Secure Computing in the Cloud

Challenges & Solutions

1. Differences in dev and production environment is exaggerated with the cloud.

Application Integration Images

2. Security is a shared responsibility in the cloud.

Namespaces

3. Sizing resources for application and managing cost efficiencies is difficult.

Control Groups

Secure Computing in the Cloud

Challenges & Solutions

Can we get all-in-one solution?

Secure Computing in the Cloud

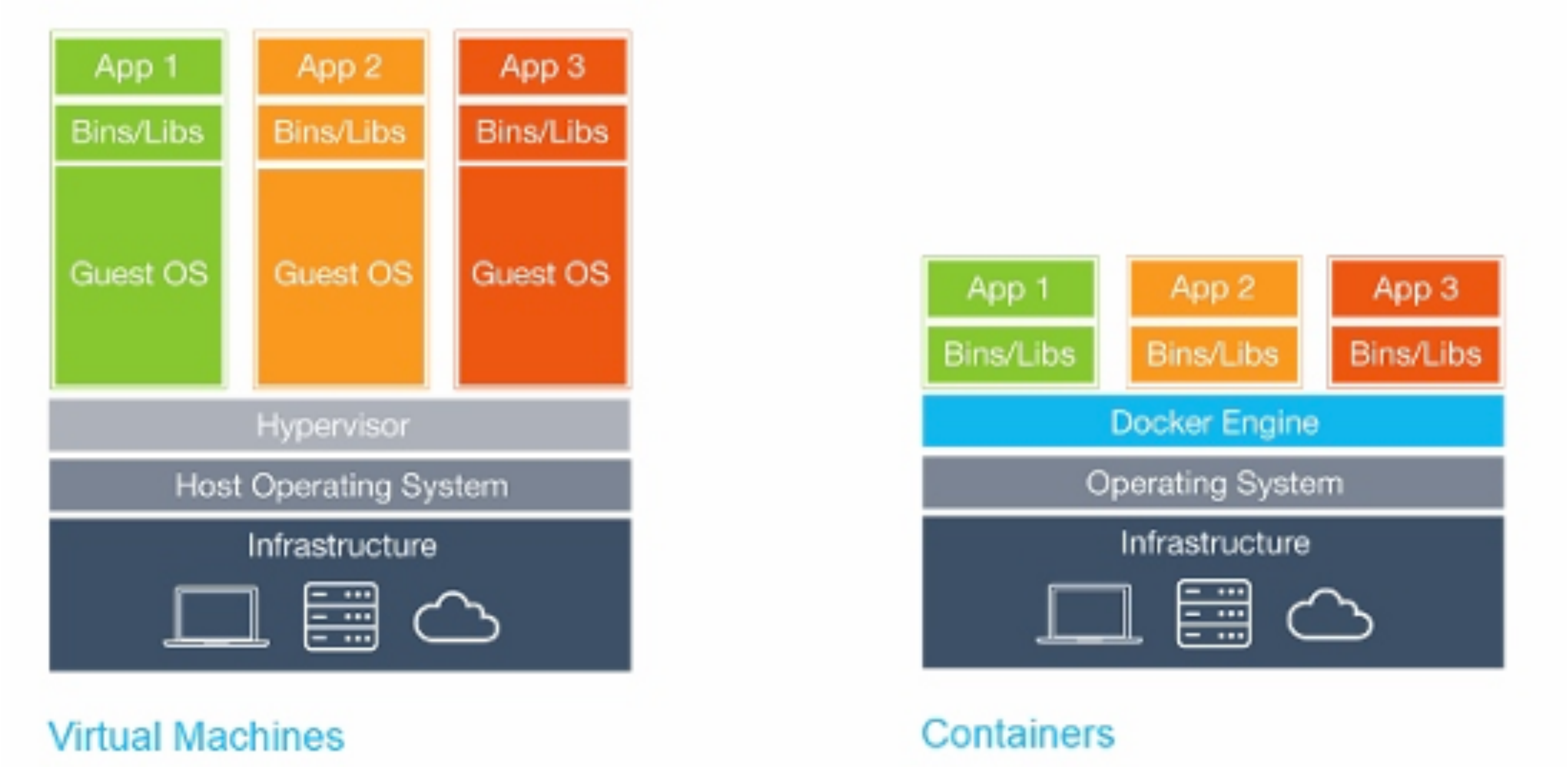
Challenges & Solutions

Can we get all-in-one solution?

Docker

Secure Computing in the Cloud

What is Docker?



Secure Computing in the Cloud

What is Docker?

- **Application Developer**: Avoid “it works on my laptop” situation.
- **Security Architect**: Immutable image and easy to use isolation technology.
- **Systems Engineer**: REST API tool to automate provisioning.
- **Ops Engineer**: Easier to kill/restart than troubleshoot.
- **Managers**: Tool to efficiently utilize the compute resources.

Secure Computing in the Cloud

Container Facts

- VM runs its own kernel, Container uses host kernel.
- Container cannot run a different OS than host
- Docker runs on VM, not on Hypervisor.
- Multiple docker containers can run inside a VM
- Vagrant Box provides vast flavors of OS images, whereas Docker Hub provides vast flavors of application images along with OS.

Secure Computing in the Cloud

Docker Terminology

- **Image:** Like a drive image of a virtual machine.
- **Dockerfile:** script that builds images
- **Layer**: Action commands in docker file commits a change like Git,
creating a new layer.
- **Registry**: Network storage of docker images
- **Docker Hub**: Library of public and private images.
- **Container**: Like a running virtual machine (just a process)

Secure Computing in the Cloud

Docker Quick Start

- Start Docker service
  - **`Cmd + space -> “Docker.app”`**
- Run Hello World
  - **`docker run hello-world`**

Secure Computing in the Cloud

Docker Quick Start

- Browse Docker images
  - **`https://hub.docker.com`**
- Start a Cent OS server and check the version
  - **`docker run centos cat /etc/os-release`**
- Start a Web server
  - **`docker run -p 80:80 nginx`**
- Start an interactive session on Cent OS server
  - **`docker run -it centos`**

Secure Computing in the Cloud

Docker Quick Start

- Start a Ubuntu server
  - **`docker run ubuntu`**
  - **`docker run -it ubuntu cat /etc/os-release`**
  - **`docker run -it ubuntu /bin/bash`**

Secure Computing in the Cloud

Docker Quick Start

- Start a Jenkins server
  - **`docker run -p 8080:8080 -p 50000:50000 jenkins/jenkins`**
  - **`http://localhost:8080`**

Secure Computing in the Cloud

Docker Quick Start

- **docker images**
- **docker ps**
- **docker rmi**
- **docker stop**
- **docker tag**
- **docker inspect -f '{{range .NetworkSettings.Networks}} {{.IPAddress}}{{end}}’ <container id>**

Check your skills

Hands-on Practice

Secure Computing in the Cloud

Containers in the Cloud

- Containerize Java Hello World
- Containerize Python Hello World
- Run a state full Postgres database (Kill and Check)
- Check your skills: Containerize a Flask Application

Secure Computing in the Cloud

CONTAINERIZE: JAVA HELLO WORLD

- `cd master-class/building-blocks/java`
- `cat Dockerfile`

```
FROM centos
RUN yum install java-1.8.0-openjdk -y
COPY HelloWorld.jar .
ENTRYPOINT ["java", "-cp", "HelloWorld.jar", “HelloWorld"]
```

- `docker build -t "helloworld" .`
- `docker history helloworld`
- `docker run helloworld`

Secure Computing in the Cloud

CONTAINERIZE: PYTHON HELLO WORLD

- `cd master-class/building-blocks/python`
- `cat Dockerfile`

```
FROM python:3
COPY dist/HelloWorld-0.1.tar.gz .
RUN pip install HelloWorld-0.1.tar.gz
CMD ["python", “/usr/local/bin/HelloWorld.py"]
```

- `docker build -t “python-helloworld" .`
- `docker history python-helloworld`
- `docker run python-helloworld`

Secure Computing in the Cloud

CONTAINERIZE: POSTGRES WITH STATE FULL DATA

- `mkdir /tmp/db`
- `docker run --rm --name postgres1 -e POSTGRES_PASSWORD=secret  -v /tmp/db:/var/lib/postgresql/data -p 5432:5432 postgres`
- `psql -h localhost -p 5432 -U postgres`

```
Password for user postgres:
postgres=# CREATE DATABASE mytest;
CREATE DATABASE
postgres=# \l
```

- Once you stop docker, what happens to data in database?

Secure Computing in the Cloud

CONTAINERIZE: WEB SERVICE WITH HEALTH CHECK

- `cd master-class/building-blocks/python`
- `cat Dockerfile`

```
FROM python:3
COPY dist/HelloWorld-0.1.tar.gz .
RUN pip install HelloWorld-0.1.tar.gz
CMD ["python", “/usr/local/bin/HelloWorld.py"]
```

- `docker build -t “python-helloworld" .`
- `docker history python-helloworld`
- `docker run python-helloworld`

Secure Computing in the Cloud

CHECK YOUR SKILLS: A FLASK APP

```
$ cat app.py
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello():
    return '<h1>Hello from Master Class!</h1>'

if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0')
```

Secure Computing in the Cloud

CHECK YOUR SKILLS: A FLASK APP

- `cd master-class/app-on-a-laptop/flask`
- `cat Dockerfile`

```
FROM python:3
COPY app.py .
RUN pip install flask
EXPOSE 5000
CMD ["python", "app.py"]
```

- `docker build -t "flask-app" .`
- `docker run -p 5000:5000 flask-app`

Secure Computing in the Cloud

(OPTIONAL) EXPLORE COMMANDS

- `Clear containers`
  - `docker rm -f $(docker ps -a -q)`
- `Clear images`
  - `docker rmi -f $(docker images -a -q)`
- `Clear volumes`
  - `docker volume rm $(docker volume ls -q)`

GOALS

- Docker Compose with NodeJS App
- Networking In-depth
- Storage In-depth
- Wordpress with Persistent Database

DOCKER COMPOSE: NODE JS & HEALTH CHECK

- cd master-class/stack-on-a-laptop/nodejs/
- cat Dockerfile

```
FROM node:latest

# environment variables
ENV dir /app
ENV port 8080

# create /app and copy files
RUN mkdir -p ${dir}
COPY . ${dir}
WORKDIR ${dir}
RUN npm install

# start chat bot service
EXPOSE ${port}
CMD ["node", "server.js"]

# health check every 30 seconds to ensure pages are served within 3 seconds
HEALTHCHECK --interval=30s --timeout=3s CMD curl -f http://localhost:8080/ || exit 1
```

DOCKER COMPOSE: NODE JS & HEALTH CHECK

- `docker build -t nodejs-image .`
- `docker run -p 8080:8080 nodejs-image`
- `docker ps # from a different window`

```
CONTAINER ID   COMMAND            CREATED              STATUS                   PORTS
9c7f93         "node server.js"   About a minute ago   Up 59 seconds (healthy)  8080/tcp
```

- `docker inspect --format "{{json .State.Health }}" 9c7f93a38e09 | jq`
- `curl` http://localhost:8080

DOCKER COMPOSE: NODE JS & HEALTH CHECK

- `cat docker-compose.yml`

```
version: '2'

services:
  n1:
    image: nodejs-image:latest
    ports:
      - “8080:8080”
```

- `docker-compose up`
- `curl http://localhost:8080`

NETWORK IN-DEPTH - 1

- `cd master-class/stack-on-a-laptop/network-indepth`
- `cat single-network.yml`

```
version: '2'
services:
  a:
    image: nginx
  b:
    image: nginx
```

- `docker-compose -f single-network.yml up`
- `docker network ls`
- `docker network inspect ${network_id}`
- `docker exec -it ${cid_1} ping -c3 ${NetName}`

NETWORK IN-DEPTH - 2

- `cd master-class/stack-on-a-laptop/network-indepth`
- `cat disjoint-network.yml`

```
version: '2'

services:
  a:
    image: nginx
    networks:
      - public

  b:
    image: nginx
    networks:
      - private

networks:
  public:
  private:
```

- `docker-compose -f disjoint-network.yml up`
- `docker network ls`
- `docker network inspect ${network_id}`
- `docker exec -it ${cid_1} ping -c3 ${NetName}`

NETWORK IN-DEPTH – 3

- `cd master-class/stack-on-a-laptop/network-indepth`
- `cat disjoint-network.yml`

```
version: '2'

services:
  a:
    image: nginx
    networks:
      - public

  b:
    image: nginx
    networks:
      - private
      - public

  c:
    image: nginx
    networks:
      - internal

networks:
  public:
  private:
  internal:
```

NETWORK IN-DEPTH - 3

- `docker-compose -f pub-priv-network.yml up`
- `docker network ls`
- `docker network inspect ${network_id}`
- `docker exec -it ${cid_1} ping -c3 ${NetName}`
- `docker exec -it ${cid_1} /bin/bash`
- `# Hint: Try hostname resolution`

STORAGE IN-DEPTH

- `cd master-class/stack-on-a-laptop/storage-indepth`
- `cat all-storage.yml`

```
version: '2'

services:
  a:
    image: nginx
    networks:
      - public
    volumes:
      - ./a.conf:/a.conf

  b:
    image: nginx
    networks:
      - private
      - public
    volumes:
      - ./b.conf.d/:/bigconfig.d/

  c:
    image: nginx
    networks:
      - internal
    volumes:
      - namedvolume:/namedvolume

  d:
    image: nginx
    networks:
      - private
    volumes_from:
      - b:ro

networks:
  public:
  private:
  internal:

volumes:
  namedvolume:
```

STORAGE IN-DEPTH

- `docker-compose -f all-storage.yml up`
- `docker exec -it ${cid_1} ls /a.conf`
- `docker exec -it ${cid_2} ls /bigconfig.d/`

- `docker exec -it ${cid_3} ls /namedvolume`
- `docker volume ls`
- `docker volume inspect ${volume_id}`

- `docker exec -it ${cid_4} ls /bigconfig.d/`
- `# try writing something in /bigconfig.d/`

CHECK YOUR SKILL

- Build a Wordpress server
  - On “frontend" subnet
- Build MariaDB as backend
  - On “backend” subnet
  - with “persistent” storage

WORDPRESS WITH PERSISTENT DATABASE

- `cd master-class/stack-on-a-laptop/wordpress`
- `cat docker-compose.yml`

```
version: '2'

services:
  wordpress:
    image: wordpress:latest
    depends_on:
      - db
    ports:
      - “8000:80"
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: wordpress
    networks:
      - frontend
    restart: always
    cpuset: "0"
    mem_limit: "100m"
```

```
  db:
    image: mariadb:10.1
    volumes:
      - "/tmp/mysqldb:/var/lib/mysql"
    environment:
      MYSQL_ROOT_PASSWORD: example
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress
    networks:
      - frontend
      - backend
    restart: always
    cpuset: "1"
    mem_limit: "200m"

volumes:
  db:

networks:
  backend:
  frontend:
```

WORDPRESS WITH PERSISTENT DATABASE

- `docker-compose up`

- `# http://localhost:8000`
- `# configure and install wordpress`
- `# publish couple of pages`
- `# Abruptly kill the infrastructure`
- `# Start it again and check`
- `# http://localhost:8000`

CHECKLIST

- Docker Compose with NodeJS App
- Networking In-depth
- Storage In-depth
- Wordpress with Persistent Database

GOAL

- Hands-on Kubernetes
  - Deploy Java Hello World
  - Deploy Flask App
  - Deploy NodeJS App
- Marathon Samples

K8 BASICS

- Cluster, Master, Nodes
- Application Deployments
- Role of containers
- Pods
- Services
- Minikube
- Kubectl

CHECK K8 INSTALLATION

- **kubectl cluster-info**
- **kubectl get nodes**

UPLOAD DOCKER IMAGES TO REGISTRY

- Tag the images to upload
  - `docker tag flask-app masterclass/flask-app`
  - `docker tag nodejs-image masterclass/nodejs-image`
  - `docker tag helloworld masterclass/java-helloworld`
- Upload the images to docker hub
  - `docker push masterclass/flask-app`
  - `docker push masterclass/nodejs-image`
  - `docker push masterclass/java-helloworld`

K8 LOCAL INSTALL

- Start minikube service
  - `minikube start`
- Open minikube dashboard
  - `minikube dashboard`
- Walk through dashboard

K8 LOCAL INSTALL

- Start the app
  - `kubectl run java-helloworld —image=masterclass/java-helloworld`
- Check output
  - `kubectl get pods`
  - `kubectl logs <podname>`

- Hint: Do not worry if the pod keeps restarting.

KUBERNETES: FLASK APP

- `cd master-class/k8-on-a-laptop/flask-app`
- `cat deployment.yml`

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: flask-app
spec:
  selector:
    name: web
    version: v0.1
  template:
    metadata:
      labels:
        name: web
        version: v0.1
    spec:
      containers:
      - name:  flask-app
        image: masterclass/flask-app
        ports:
        - containerPort: 5000
```

- `cat service.yml`

```
apiVersion: v1
kind: Service
metadata:
  name: flask-app
spec:
  type: LoadBalancer
  ports:
    - port: 5000
      targetPort: 5000
  selector:
    name: web
```

KUBERNETES: FLASK APP

- Start deployment and service
  - `kubectl apply -f deployment.yml -f service.yml`
- Check output
  - `kubectl get pods`
  - `minikube service flask-app`

CHECK YOUR SKILL

- Deploy Node JS app with K8

KUBERNETES: NODEJS APP

- cd master-class/k8-on-a-laptop/nodejs-app
- cat deployment.yml

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: nodejs-app
spec:
  selector:
    name: web
    version: v0.1
  template:
    metadata:
      labels:
        name: web
        version: v0.1
    spec:
      containers:
      - name:  nodejs-app
        image: masterclass/nodejs-image
        ports:
        - containerPort: 8080
```

- cat service.yml

```
apiVersion: v1
kind: Service
metadata:
  name: nodejs-app
spec:
  type: LoadBalancer
  ports:
    - port: 8080
      targetPort: 8080
  selector:
    name: web
```

KUBERNETES: NODEJS APP

- Start deployment and service
  - `kubectl apply -f deployment.yml -f service.yml`
- Check output
  - `kubectl get pods`
  - `minikube service nodejs-app`

CHECKLIST

- Hands-on Kubernetes
    - Deploy Java Hello World
    - Deploy Flask App
    - Deploy NodeJS App

What is due?

Homework & Discussions