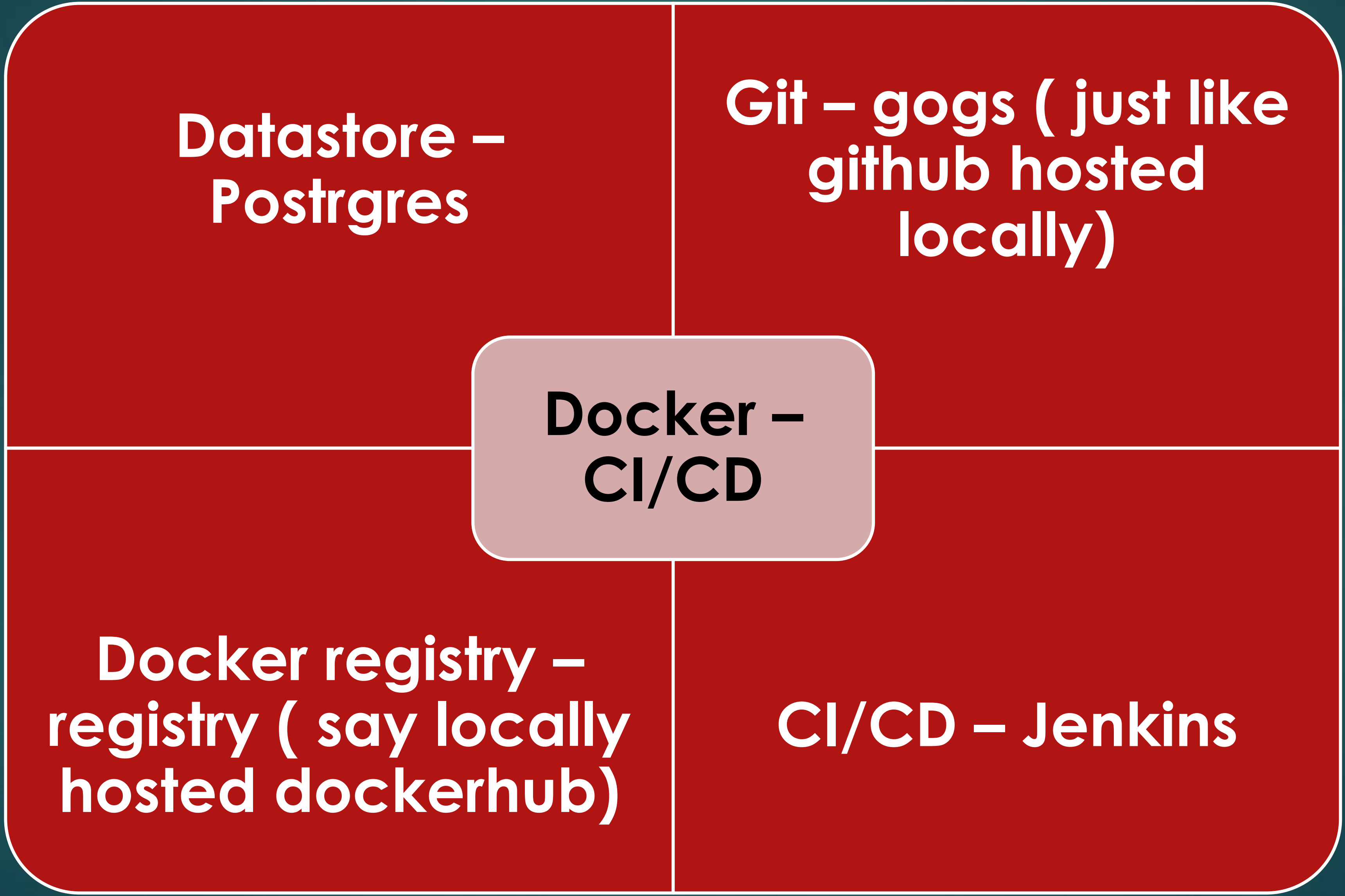
Docker – CI/CD


Datastore – Postrgres
Git – gogs (just like github hosted locally)
Docker – CI/CD
Docker registry – registry (say locally hosted dockerhub)
CI/CD – Jenkins

Follow Along Guide

Textual Slides

A Note for Powershell Users

Terminal commands reflect the Unix bash shell. PowerShell users will need to adjust the commands.

- Unix Variables
  - `export MY_VAR=test`
  - `echo ${MY_VAR}`
- Windows 10 Variables (powershell)
  - `$env:my_var = "test"`
  - `Get-ChildItem Env:my_var`

Docker in Translation

- **Docker client**
  - The docker command used to control most of the Docker workflow and talk to remote Docker servers.
- **Docker server**
  - The dockerd command used to launch the Docker daemon. This turns a Linux system into a Docker server that can have containers deployed, launched, and torn down via a remote client.

Docker in Translation

-
  - Virtual Machine
    - In general, the docker server can be only directly run on Linux. Because of this, it is common to utilize a Linux virtual machine to run Docker on other development platforms. Docker Community/Desktop Edition makes this very easy.

Docker in Translation

- **Docker images**
  - Docker images consist of one or more filesystem layers and some important metadata that represent all the files required to run a Dockerized application. A single Docker image can be copied to numerous hosts. A container will typically have both a name and a tag. The tag is generally used to identify a particular release of an image.

Docker in Translation

- **Linux Containers**
  - A Linux Container is a single instantiation of a Docker or OCI-standard image. A specific container can only exist once; however, you can easily create multiple containers from the same image.
  - OCI - Open Container Initiative

Docker Engine isn’t a…

- virtualization platform (VMware, KVM, etc.)
- cloud platform (AWS, Azure, etc.)
- configuration management tool (Chef, Puppet, etc.)
- deployment framework (Capistrano, etc.)
- development environment (Vagrant, etc.)
- workload management tool (Mesos, Kubernetes, etc.)

Linux Namespaces

- Mount (filesystem resources)
- UTS (host & domain name)
- IPC (shared memory, semaphores)
- PID (process tree)
- Network (network layer)
- User (user and group IDs)

Control Groups (cgroups)

- Resource limiting
- Prioritization
- Accounting
- Control

Setting the Stage

```
$ cd ${HOME}
$ mkdir class
$ cd class
$ mkdir code
$ git clone https://github.com/chirag99969/dockercicd.git
layout --config core.autocrlf=input
$ cd layout
$ ls
```

Automating Workflow

- Datastore
  - Postgres
- Collaborative Source Code Repository
  - Gogs
- Docker Image Repository
  - Docker Distribution
- Build, Test, and Deploy
  - Jenkins

Docker – CI/CD

Datastore – Postrgres

Git – gogs (just like github hosted locally)

Docker – CI/CD

Docker registry – registry (say locally hosted dockerhub)

CI/CD – Jenkins

Iterative Workflow

- Core Technology - Docker

User develops code locally (Docker)
User commits code (Gogs backed by Postgres)
Pipeline builds & tests code (Jenkins & Docker Distribution)
Pipeline deploys code to production.
and then iterate...

Composing a Docker Service

- Open & explore docker-compose.yaml in your text editor
- Full Documentation:
  - https://docs.docker.com/compose/compose-file/

Creating a Datastore

```
$ cd compose/review/1st
$ vi docker-compose.yaml
```

- Note: DB user & password

Creating a Source Repo

```
$ cd ../2nd
$ vi docker-compose.yaml
```

Docker Distribution

```
$ cd ../3rd
$ vi docker-compose.yaml
```

Manage Secrets

```
$ cd ../../final
$ echo 'MY_PG_PASS=myuser-pw!' > ./.env
```

- On Windows try:
  - `Add-Content ./.env "MY_PG_PASS=myuser-pw!"`

Jenkins

```
$ vi docker-compose.yaml
$ docker compose config
$ docker compose up -d
$ docker compose ps
$ docker compose logs -f
2017/07/01 20:06:31 [ INFO] Listen: http://0.0.0.0:3000
LOG: database system is ready to accept
connections msg="debug server listening
localhost:5001"
Please use the following password to proceed to installation
```

Configure Gogs

- Navigate web browser to:
  - http://127.0.0.1:10090/install
- **Database Type**: `Postgresql`
- **Host**: `postgres:5432`
- **User**: `postgres`
- **Password**: `myuser-pw!`
- **SSH Port**: `22`
- **Application URL**: `http://3.84.164.83:10090/`

Create Gogs User

- Click: **Admin Account Settings**

Username: `myuser`
Password: `myuser-pw!`
Confirm Password: `myuser-pw!`
Email Address: `myuser@example.com`
Click: `Install Gogs`

Create GIT Repo

Click: `+`
Click: `+ New Repository`
Repository Name: `outyet`
Click: `Create Repository`

Git Credentials

- In the next section **Windows users** will likely see a GUI based password prompt from git.
- **Unix users** will likely just see a text based prompt.
- Be sure to provide your gogs username and password for the prompt.

Explore the Code

```
$ cd ~/class/code/outyet
```

- Explore with your favorite code editor
  - Dockerfile
  - main.go
  - main_test.go

```
docker compose up -d
```

Examine Application

- Navigate web browser to:
  - http://127.0.0.1:10088/

```
$ docker compose down
```

First Code Commit

```
$   cd   ../../..
$   cp   -a outyet ../code/
$   cd   ../code/outyet/
$   git   init
$   git   config core.autocrlf input

$   git   add .
$   git   commit -m "first commit"
```

- **Note**: At the moment `Gogs` expects a branch named `master`, instead of the newer `main` standard.

Push Upstream

```
$ git remote add origin http://localhost:10090/myuser/outyet.git
$ git push -u origin master
```

- username: `myuser`
- password: `myuser-pw!`

Docker Distribution Login

```
$ docker login 127.0.0.1:5000
```

- username: `myuser`
- password: `myuser-pw!`

NOTE: The example registry TLS certificate includes a SAN for `private-registry.localdomain`. You can add an entry in `/etc/hosts` or `C:\Windows\System32\Drivers\etc\hosts` to point this domain name at a remote IP address if needed.

Test Docker Distribution

```
$ docker image pull cybersecnerd/unsc_infinity:gravemind
$ docker image ls cybersecnerd/unsc_infinity:gravemind
$ docker image tag ${IMAGE_ID} 127.0.0.1:5000/myuser/unsc_infinity:gravemind
$ docker image push 127.0.0.1:5000/myuser/unsc_infinity:gravemind
```

Configure Jenkins

```
cat ../../layout/jenkins/data/secrets/initialAdminPassword
```

- Navigate web browser to:
  - http://127.0.0.1:10091/

- Paste Administrator Password

  Click: `Continue`

  Click: `Select plugins to install`

  Click: `None`

  Click: `Install`

Configuring Jenkins

- Create Admin User

  Username: `myuser`

  Password: `myuser-pw!`

  Confirm password: `myuser-pw!`

  Full Name: `My User`

  E-Mail Address: `myuser@example.com`

  Click: `Save and Continue`

Configuring Jenkins

- Final Details

  Jenkins URL: `http://127.0.0.1:10091/`

  Click: `Save and Finish`

  Click: `Start Using Jenkins`

Shutdown Services

```
$ cd ~/class/layout/compose/final
$ docker compose stop
```

Getting Started with Jenkins

- Navigate web browser to:
  - http://127.0.0.1:10091/
- Login to Jenkins

  Click: `create new jobs`

Note: If you have not configured Jenkins, you can login using the `admin` user and the `initialAdminPassword`.

Creating The Jenkins Job

Enter an item name: `outyet`
Click: `Freestyle project`
Click: `OK`

Configuring the Job

Description: `build and test outyet`

Gogs Webhook

- Gogs Secret: `12345`

Source Code Management

Select: `git`
Repository URL: http://gogs:3000/myuser/outyet.git
Branch Specifier (blank for 'any'): ``

Build Triggers

- None

Build Environment

Check: `Delete workspace before build starts`
Check: `Mask passwords and regexes`
Name/Password Pairs:

- **Name**: `DOCKER_PW`
- **Password**: `myuser-pw!`

The Build Script

- Select 'Execute Shell'

```
# This is not ideal, but reasonable for class.
echo "${DOCKER_PW}" | docker login --username=myuser \
    --password-stdin 127.0.0.1:5000
docker image build -t 127.0.0.1:5000/myuser/outyet:${GIT_COMMIT}
. docker image push 127.0.0.1:5000/myuser/outyet:${GIT_COMMIT}
```

Post-Build Actions

- None

Click: Save

Build The Code

Click: `Build Now`

Build The Code

Click: `#1`

Click: `Console Output`

Build Results

Looking for:

```
Finished: SUCCESS
```

Components Assembled

- Postgres Database
  - https://www.postgresql.org/
- Gogs - Source Code Manager
  - https://gogs.io/
- Docker Distribution
  - https://github.com/docker/distribution
- Jenkins CI
  - https://jenkins.io/

What We Have Learned

- Docker Compose
- Building / Running
- Ports, Volumes, and Networks
- Launched and configured:
  - Postgres / Gogs
  - Docker Distribution
  - Jenkins