Component-based Software Development

Intro to Microservices, DevOps, Containers, and Container Orchestration

Dr. Vinod Dubey
SWE 645
George Mason University

ACKNOWLEDGEMENT

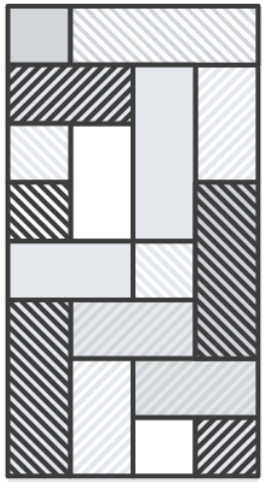
Information on some of the slides are adapted from materials on

- **http://aws.amazon.com**
- **https://linuxize.com/post/how-to-install-jenkins-on-ubuntu-20-04/**
- **https://www.jenkins.io/doc/**
- **https://hub.docker.com/**

Microservices

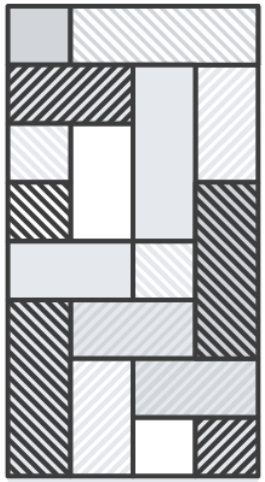
Problem

- **Monolithic Systems**
  - There used to be the case when companies will build monolithic systems
  - All in one approach
    - Most capabilities of the application with presentation, business logic, and data access packaged together
    - Run as one big executable with several jars and wars, in one code base
  - Adds complexity
    - Tightly coupled; hard to change
  - Longer development lifecycle
  - Longer testing life cycle
    - (e.g., regression testing of the entire application even for a small change in the code base)
  - Slow software delivery process
  - Inflexible Scalability – via increasing the instance size or by duplicating the instance
    - Requires scaling the entire package



Problem

- **Monolithic Systems**
  - There used to be the case when companies will build monolithic systems
  - All in one approach
    - Most capabilities of the application with presentation, business logic, and data access packaged together
    - Run as one big executable with several jars and wars, in one code base
  - Adds complexity
    - Tightly coupled; hard to change
  - Longer development lifecycle
  - Longer testing life cycle
    - (e.g., regression testing of the entire application even for a small change in the code base)
  - Slow software delivery process
  - Inflexible Scalability – via increasing the instance size or by duplicating the instance
    - Requires scaling the entire package
- **Solution: Microservices**

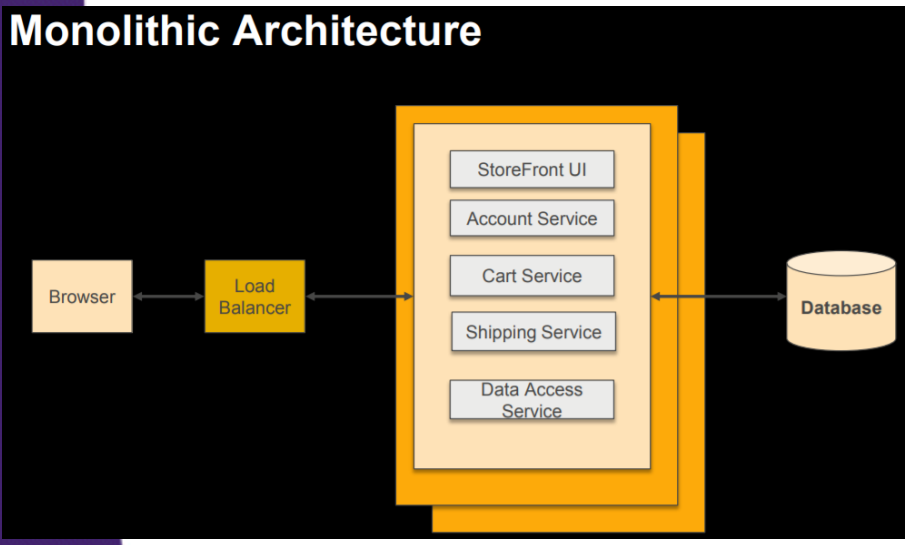


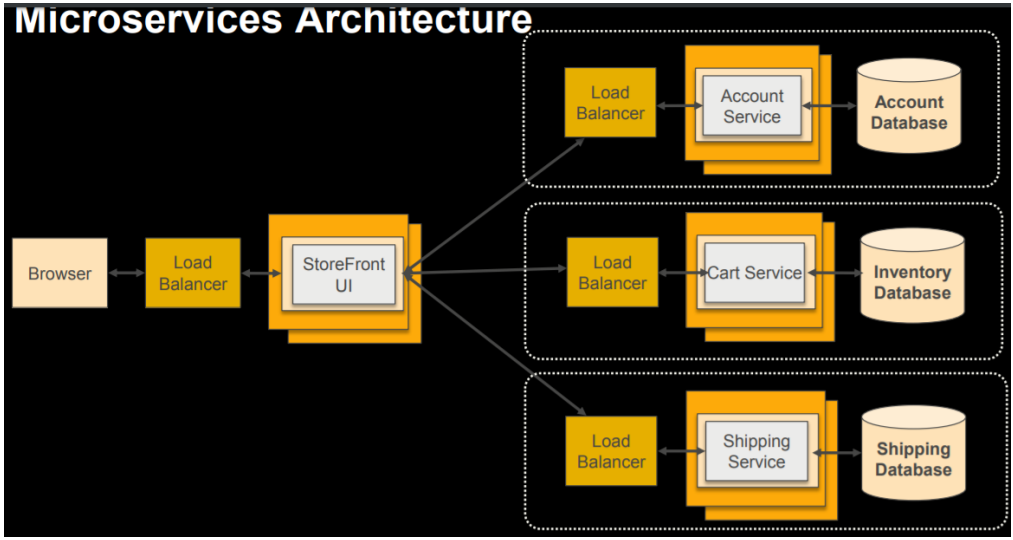
Microservices

- **A software architecture style to build modular applications**
- **Complex applications are split into relatively small, independent and loosely coupled applications**
  - That can be deployed independently and
  - That can be scaled independently

Monolithic vs. Microservices

- **Essentially, instead of having one big monolithic application that does everything, you split it up into several smaller applications**



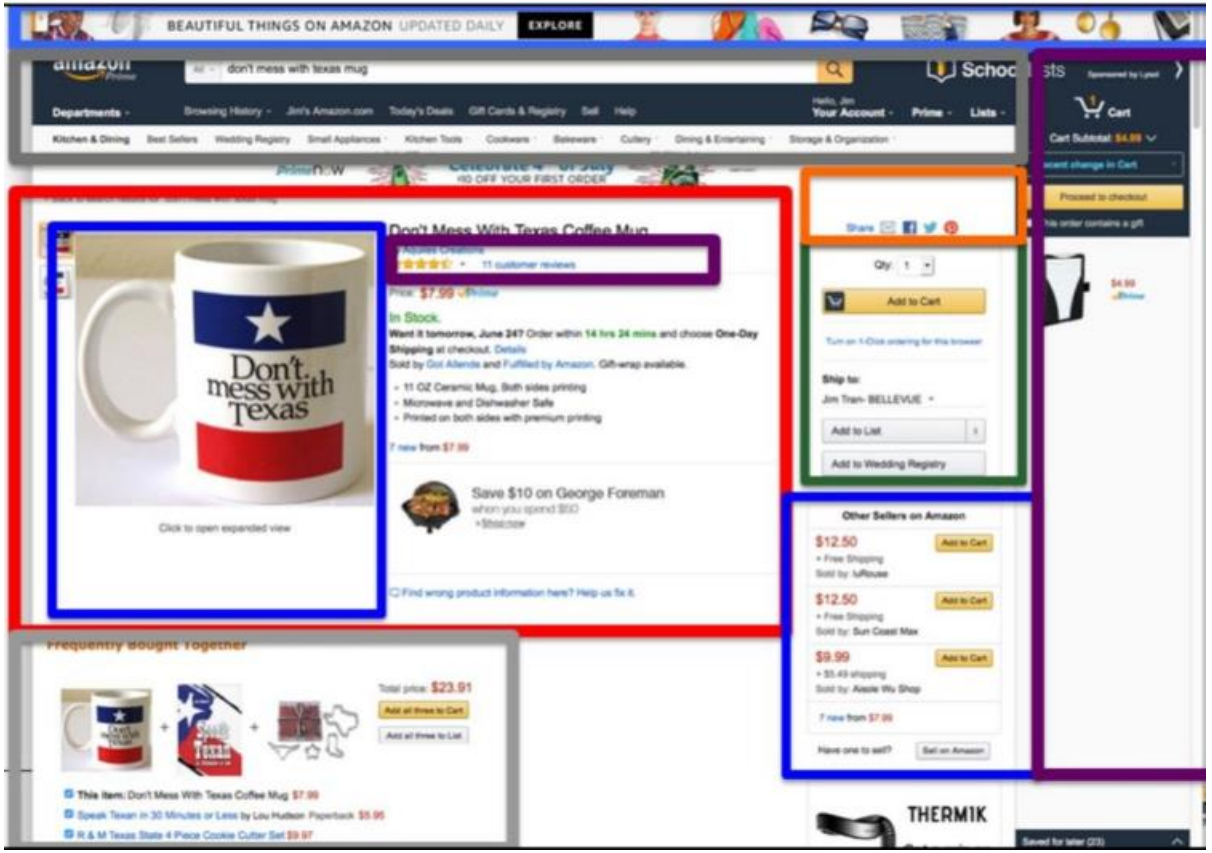


Microservices

- **Amazon.com is built using Microservices**
  - Different components of the page are built using individual Microservices - as its own individual applications

Microservices

- **Amazon.com is built using Microservices**
  - Different components of the page are built using individual Microservices - as its own individual applications



Characteristics of Microservices

- **Organized around business capabilities**
  - Authentication/Authorization, logging, shopping cart, ordering, billing, inventory, or user management
- **Services communicate using technology-agnostic protocols, such as HTTP**
- **Supports polyglot programming and polyglot persistence**
  - Implement different capabilities using different programming languages, databases, hardware and software environment, depending on what fits the best to implement the requirement.

Characteristics of Microservices

- **Independently deployable, scalable, and elastic**
  - Makes is easy to expand or shrink the components to match the workload
- **Microservices-based applications are ideal to be containerized**
  - Containerizing a microservice makes it really scalable, resilient, fault tolerant using container orchestration platforms
    - This is one of the biggest advantages of microservices!
- **Loosely coupled**
  - Updating one service does not affect other pieces of the application; Standard-based well-defined APIs to interact with

Characteristics of Microservices

- **Services are small in size, bounded by contexts**
  - Self contained
  - Follows the Unix philosophy of "Do one thing and do it well“
- **Released with automated processes,**
  - such as DevOps with independent CI/CD
  - Allows building, testing, deploying self-contained services independently without waiting for other projects/services

Key Advantages of Microservices

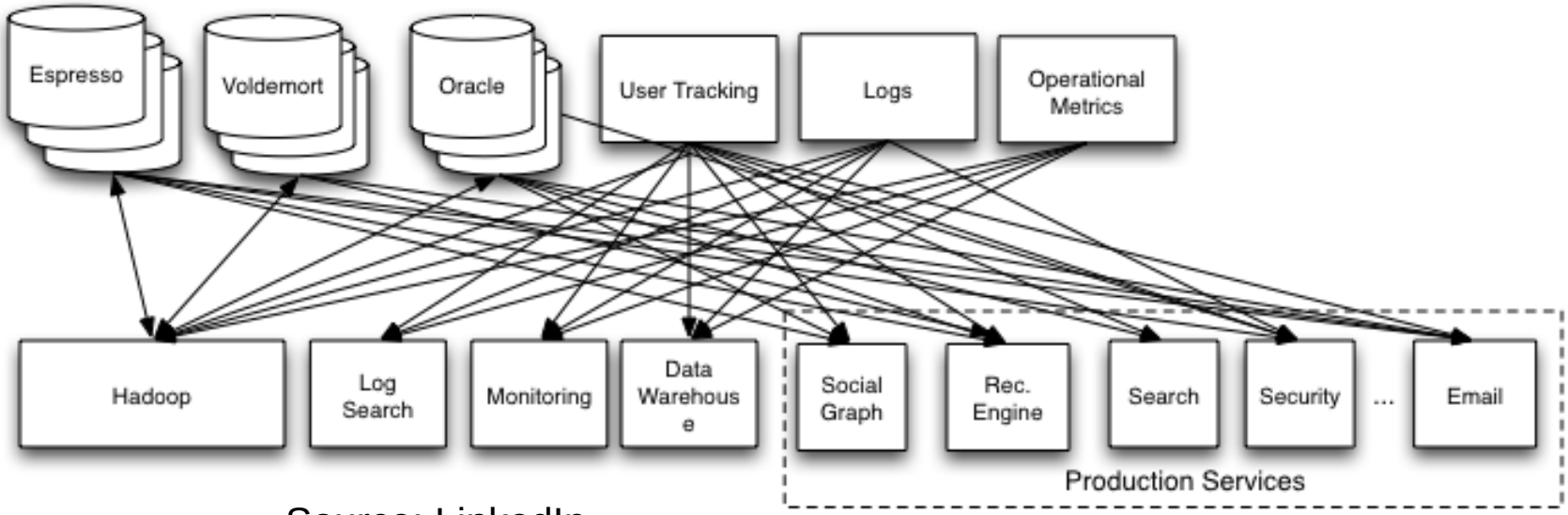
- **Language independence to implement individual microservices**
  - Since Microservices communicate via HTTP, the language used to implement the backend is irrelevant.
  - No concern integrating/interoperating between microservices implemented in Java, .NET, Python, or GO
- **Smaller teams responsible for the lifecycle of each microservice**
  - In microservices architecture, usually three or four people get assigned on each microservice – numbers may vary
  - A team is responsible specifically for that microservice and everything it does.
- **Faster iteration/Independent DevOps Pipeline**
  - The individual teams can pick the iteration cycle that work the best for them
  - Enables teams to work efficiently and easily on very specific business function

Key Advantages of Microservices (Contd.)

- **Fault and exception isolation inside your program**
  - If one microservice fails, that doesn't necessarily mean the entire application fails, especially if you are practicing RESTful APIs
    - it doesn't have to essentially crash the whole system.
- **Designed to work well with containers, and scalable**
  - Easy to scale up and down if you don't need as many instances running

Side affect of Microservices

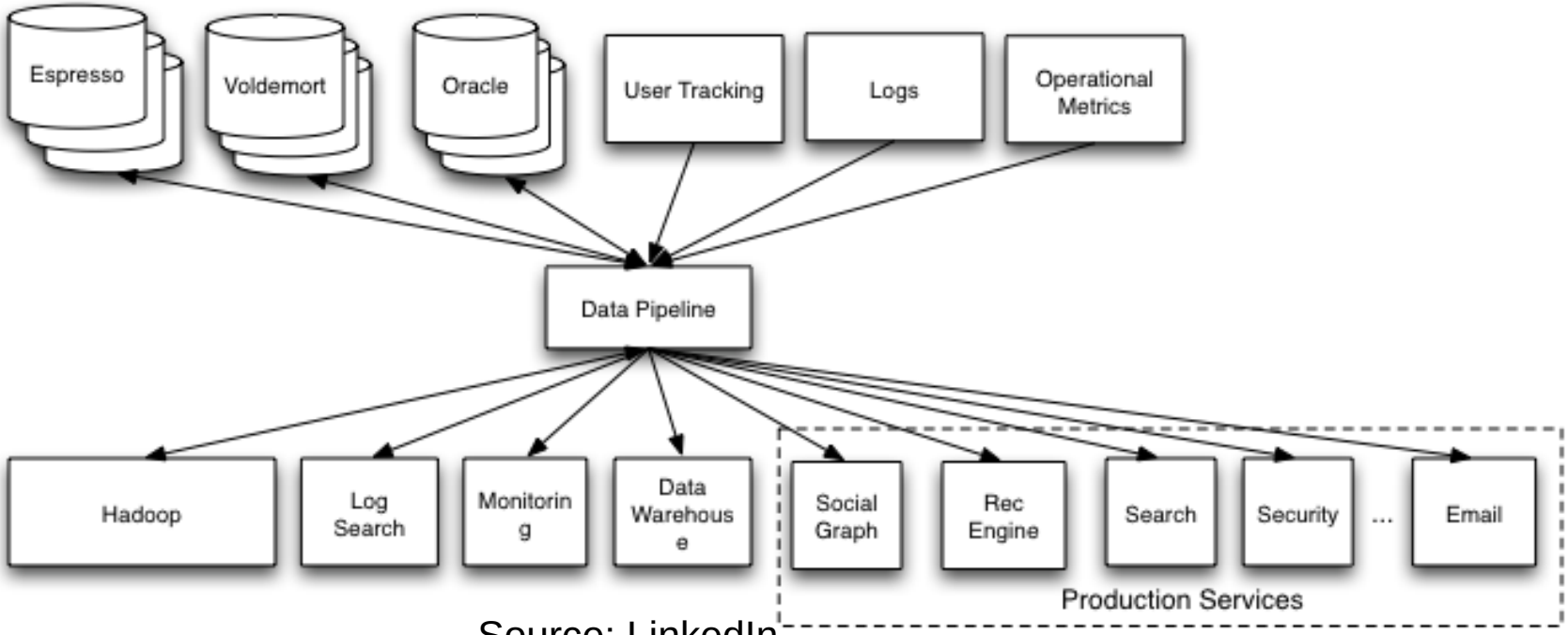
- **Issues**
  - Sprawl of microservices has potential of creating too many point to point connections,
    - that may be difficult to maintain and difficult to scale
  - Synchronous communications over http may not be performance friendly, especially under heavy workload



Source: LinkedIn

Side affect of Microservices

- **Solution**
  - Event driven microservices
  - Leverage synchronous as well as asynchronous communication
  - *More on this during Kafka lecture discussion*



Source: LinkedIn

DevOps

Issues with Traditional Software Development Approaches

- **Longer development cycles**
  - Slower innovation
- **Increased deployment failures, rollbacks, and time to recover**
- **Inadequate communication and collaboration**
- **Decreased efficiencies**
- **Increased costs and IT headcount**

Issues with Traditional Software Development Approaches

- **Longer development cycles**
  - Slower innovation
- **Increased deployment failures, rollbacks, and time to recover**
- **Inadequate communication and collaboration**
- **Decreased efficiencies**
- **Increased costs and IT headcount**

- **Solution: DevOps**

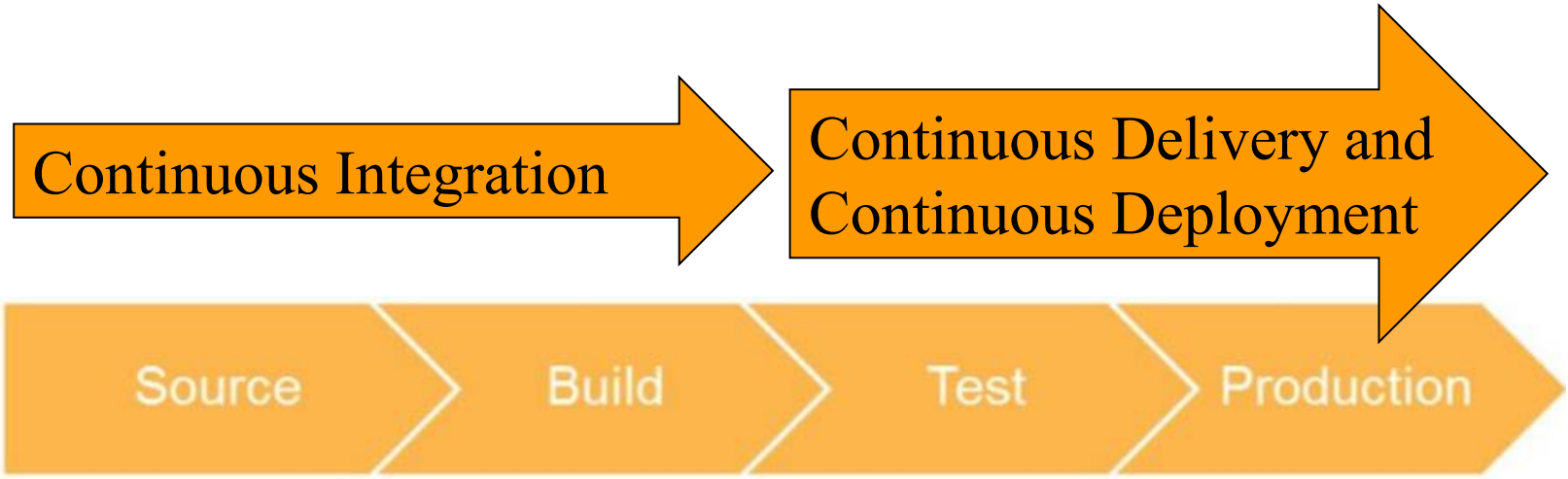
DevOps

- **A software development and delivery process that emphasizes communication and collaboration**
  - between product management, software development, and operations professionals
- **Focuses on automation and monitoring**
  - the process of software integration, testing, deployment, and infrastructure changes
- **Establishes a culture and environment** where **building, testing, and releasing** software can happen **rapidly, frequently**, and more reliably.

Source: https://en.wikipedia.org/wiki/DevOps

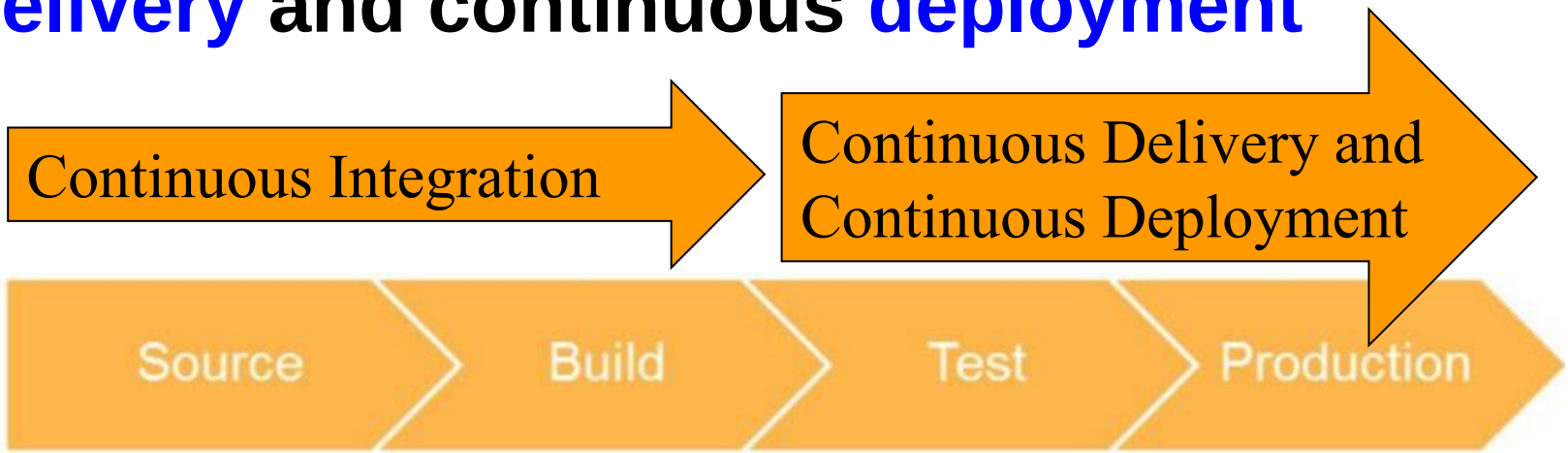
DevOps

- **Enhanced collaboration**
  - of Development and Operations teams
- **Automation**
  - of build, test, and deployment processes
- **Facilitates continuous integration, continuous delivery, and continuous deployment (CI/CD)**



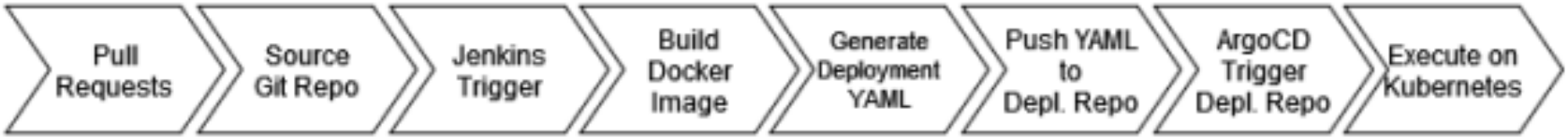
DevOps

- **Enhanced collaboration**
  - of Development and Operations teams
- **Automation**
  - of build, test, and deployment processes
- **Facilitates continuous integration, continuous delivery and continuous deployment**



DevOps

- **Typical CI/CD Pipeline using Git, Jenkins, Argo CD, and Kubernetes**
  - Merge/push your code into the source Git repo.
  - Setup Jenkins to repeatedly poll for changes on this repository, and when it detects a change, it executes a new build job that pulls the source code.
  - The docker image is built based on the Dockerfile present in the code and is pushed to the docker hub.
    - A tag is attached to this docker image based on the jenkins build number
      - IMAGE=rdlgmu/survey-frontend:$BUILD_NUMBER
      - docker build . -t $IMAGE
      - docker push $IMAGE



DevOps Tools: Git

- **Source Code Repository**
- **"Git is a distributed version-control system**
  - for tracking changes in source code during software development.
- **Designed for coordinating work among programmers, and to track changes in any set of files." Wikipedia**

A command-line session showing repository creation, addition of a file, and remote synchronization

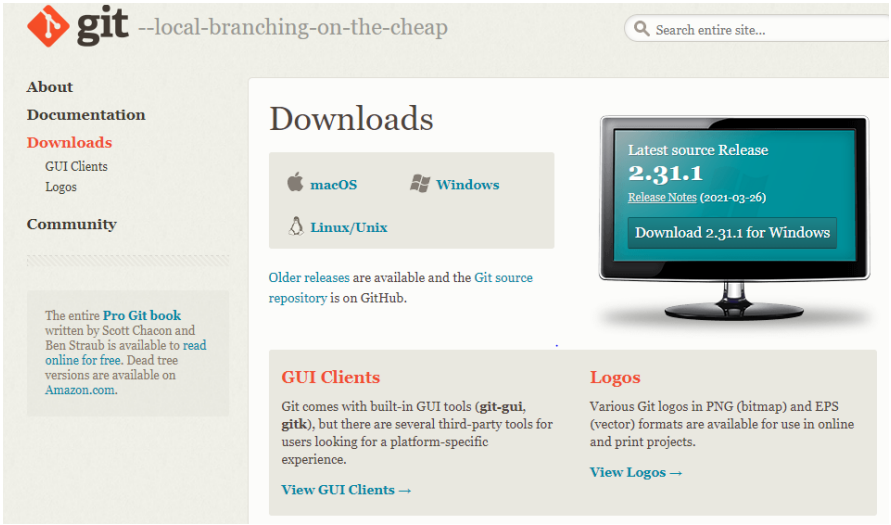
DevOps Tools : Git

- **Git can easily integrate with DevOps workflow by hosting repositories**
  - where the team members share their work.
- **Two best Git repository hosting services are GitHub and BitBucket**
  - Both of them integrate well with other DevOps tools.

A command-line session showing repository creation, addition of a file, and remote synchronization

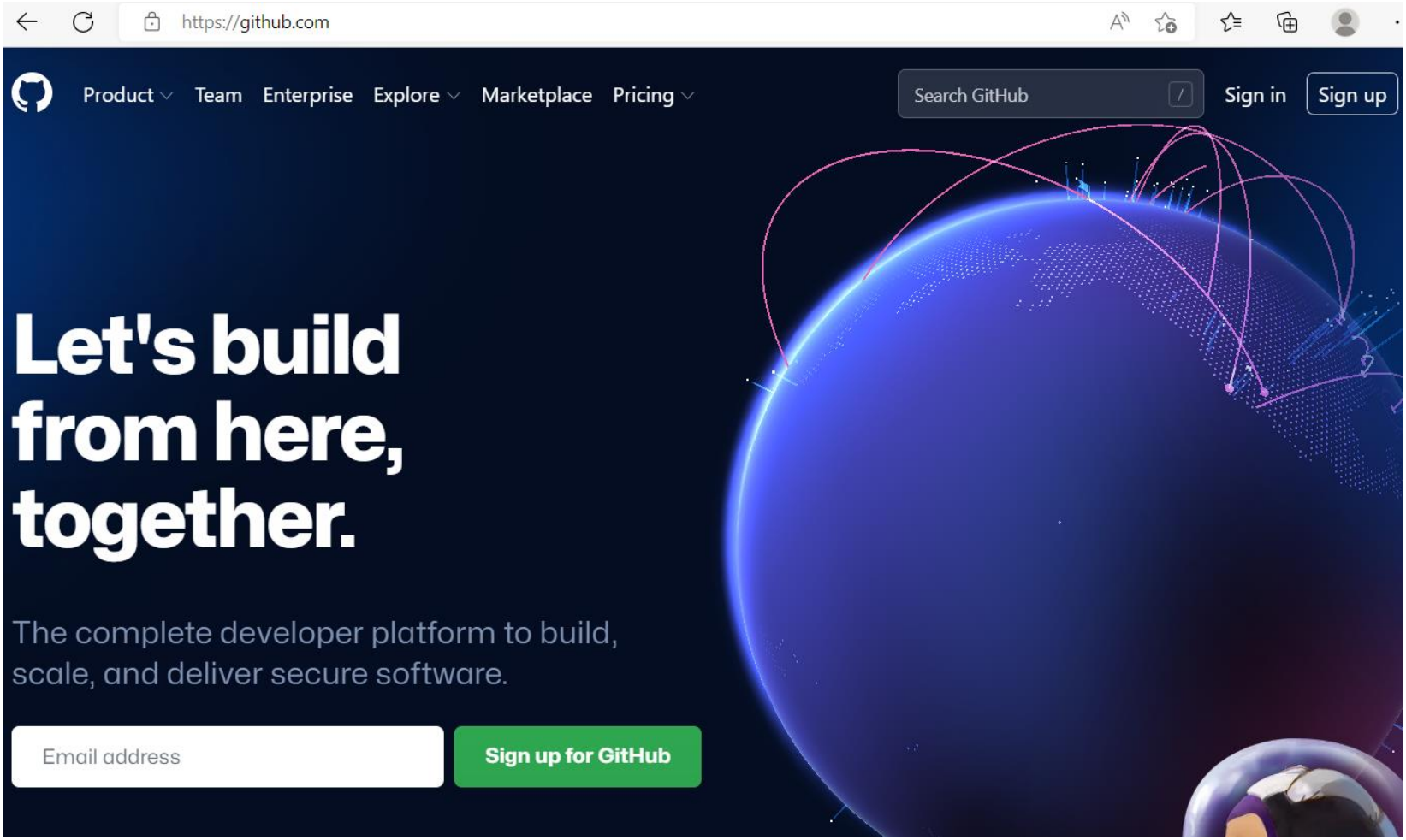
DevOps Tools : Git

- **Git maybe pre-installed on Mac machines**
- **You can install Git Bash on windows**
  - A useful link: https://www.youtube.com/watch?v=qdwWe9COT9k
    - Bash
      - Basically an Emulator for running Unix (Linux) shell on Windows
    - Git
      - Version Control Software that helps developers collaborate when building software and websites
        - https://github.com/
    - Git Bash is essentially a package that installs both Git and Bash at the same time
      - https://git-scm.com/downloads



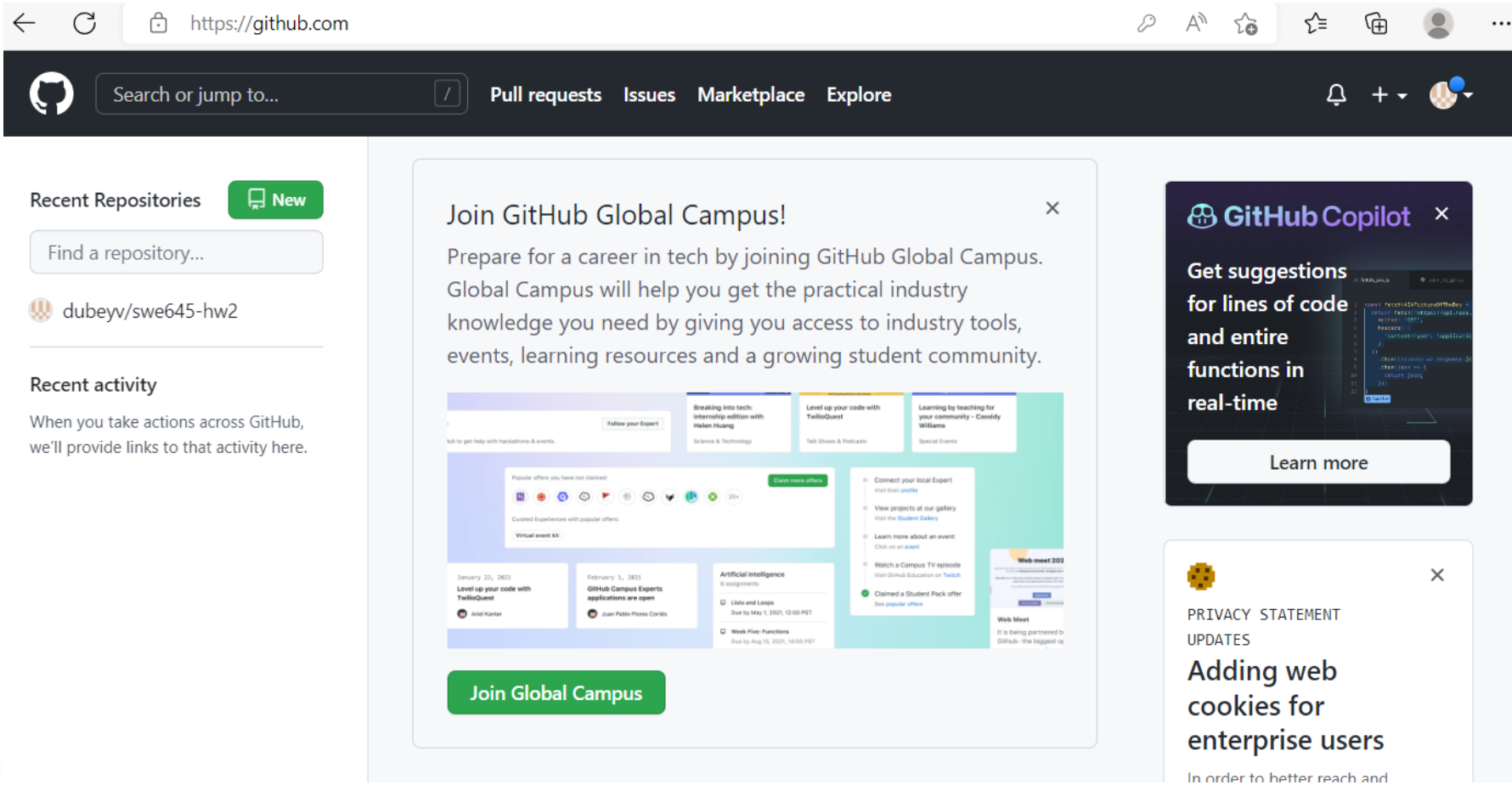
DevOps Tools : Git

- **Create an account and login to github.com**



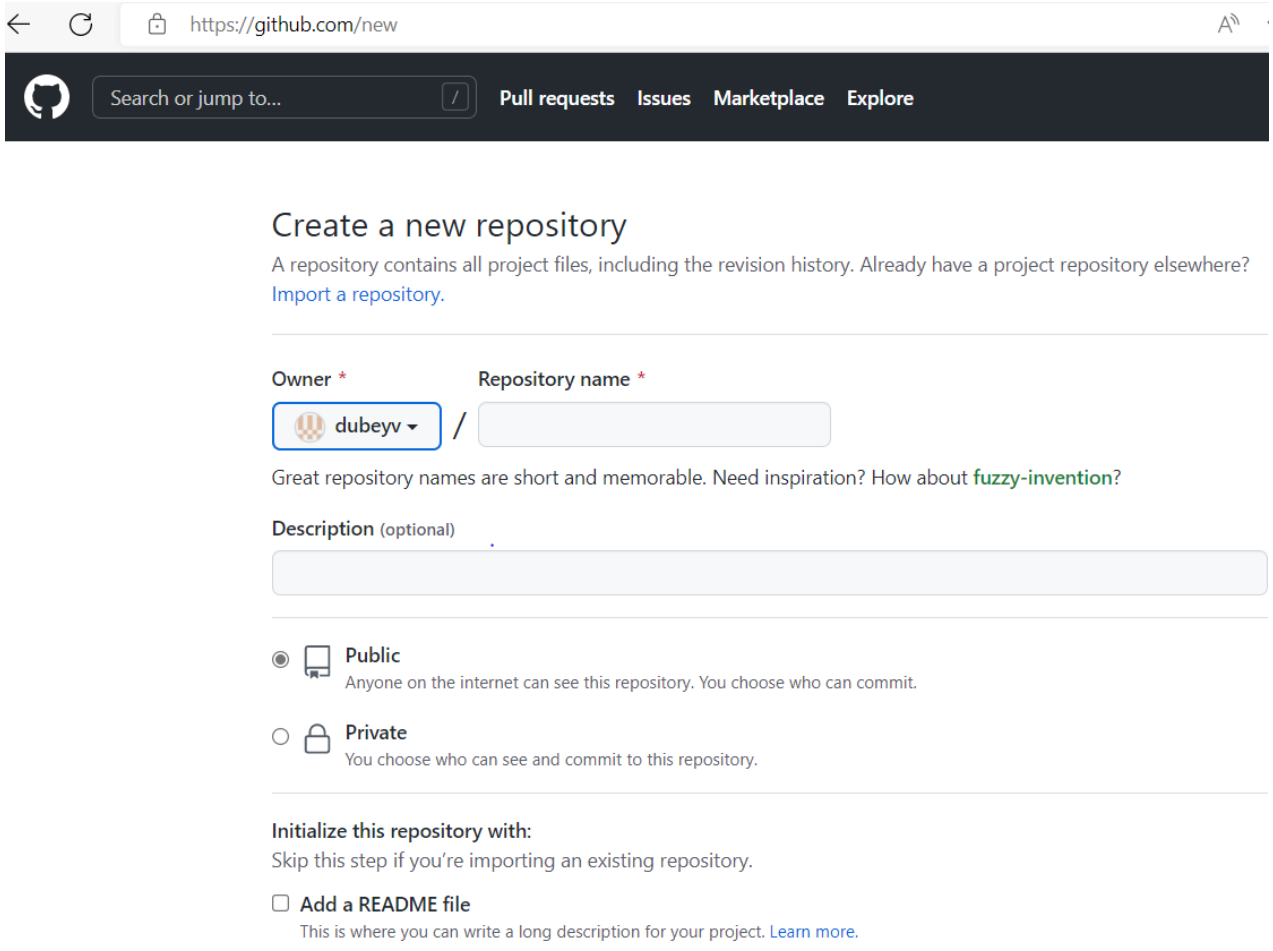
DevOps Tools : Git

- **Create an account and login to github.com**
  - Click on New to create a new repository



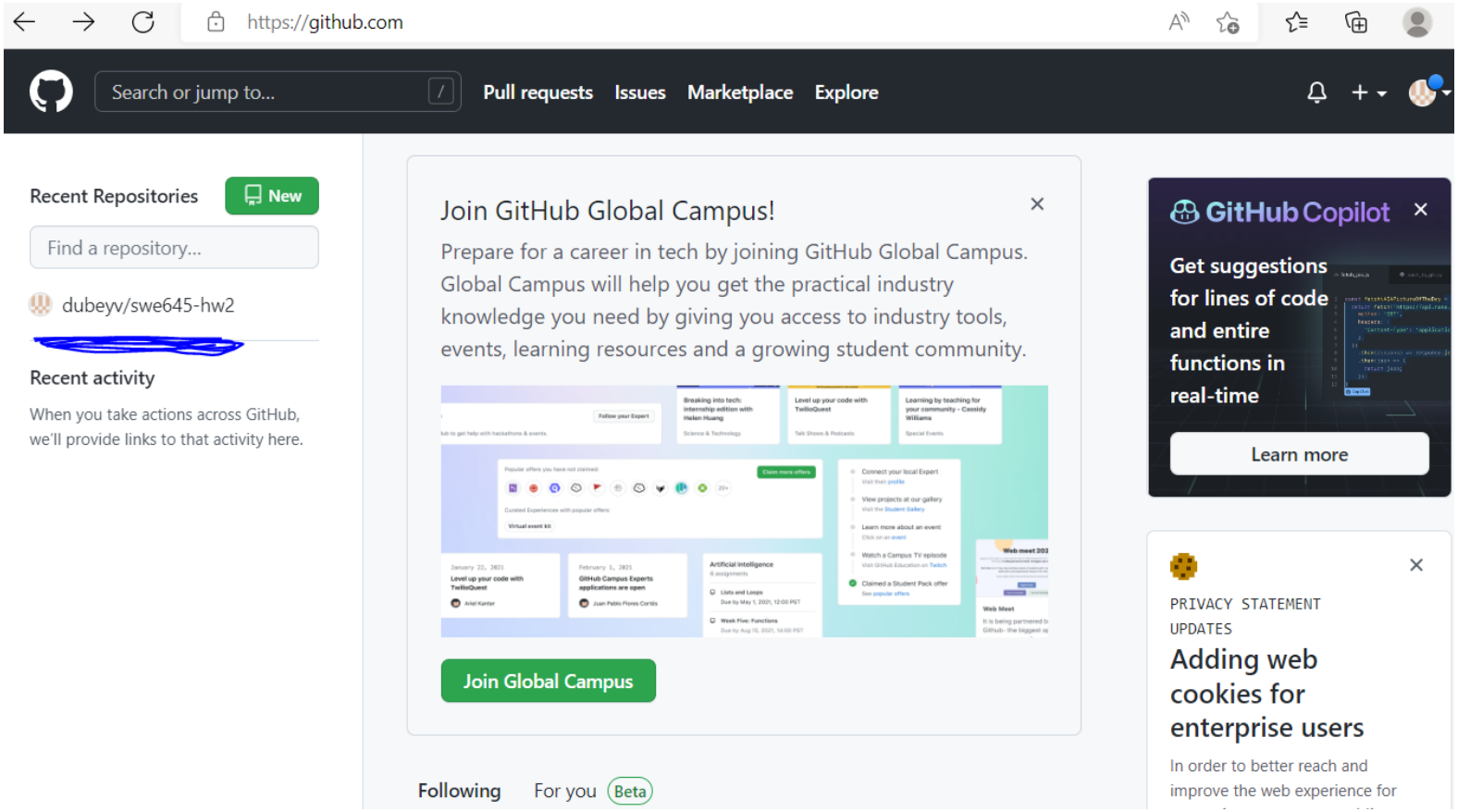
DevOps Tools : Git

- **Create an account and login to github.com**
  - Click on New to create a new repository



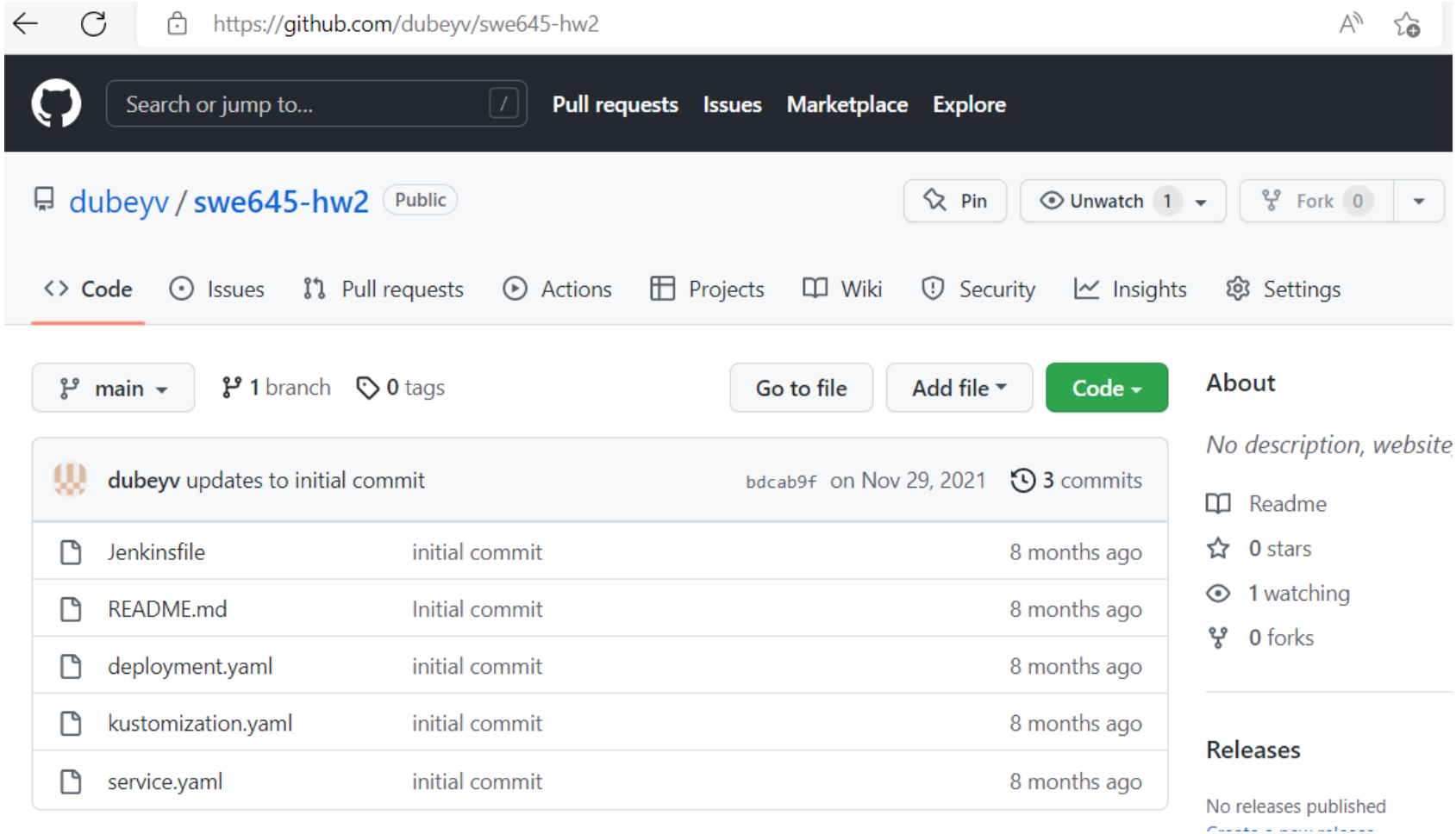
DevOps Tools : Git

- **Create an account and login to github.com**
  - Click on New to create a new repository



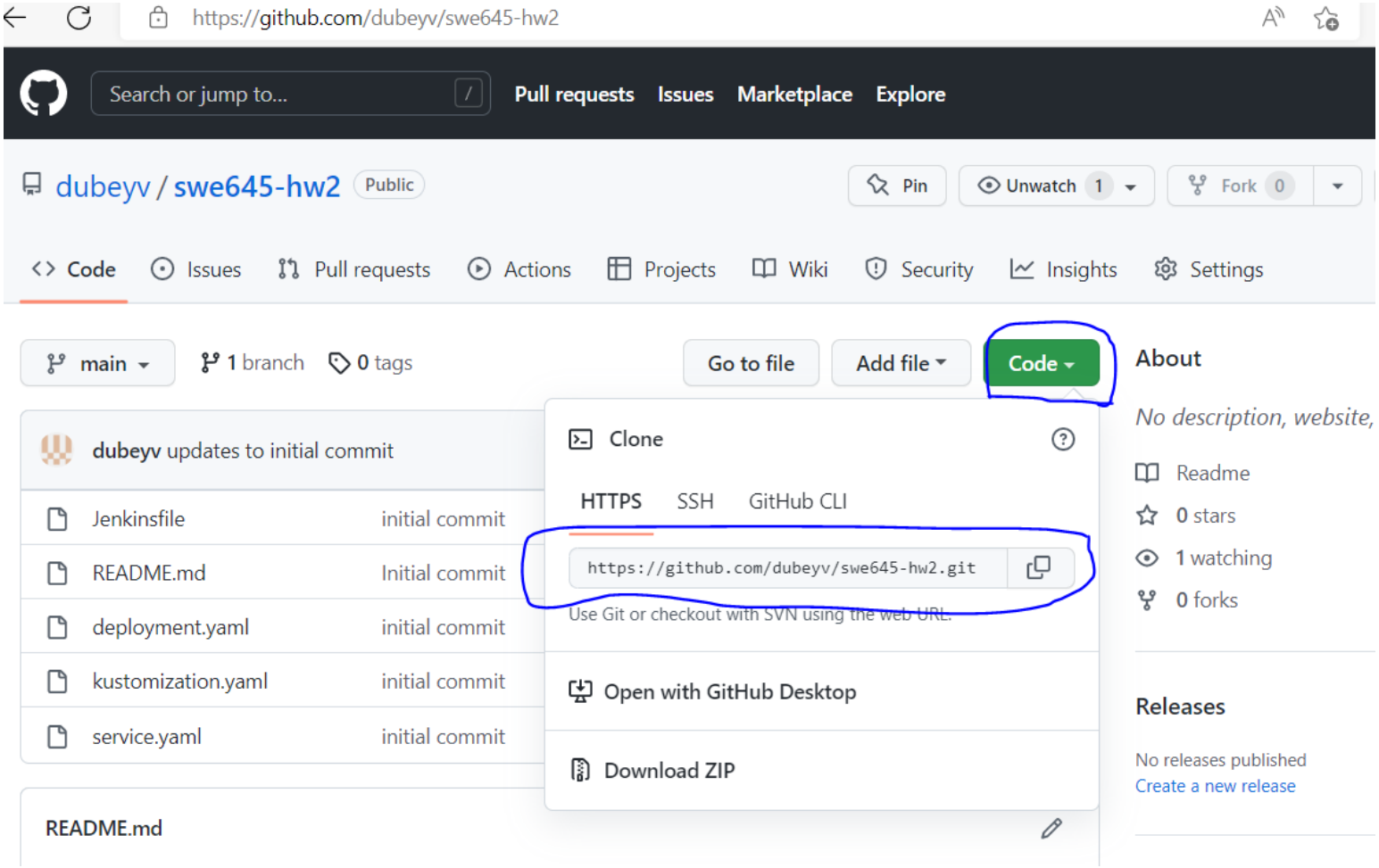
DevOps Tools : Git

- **Create an account and login to github.com**
  - Click on New to create a new repository



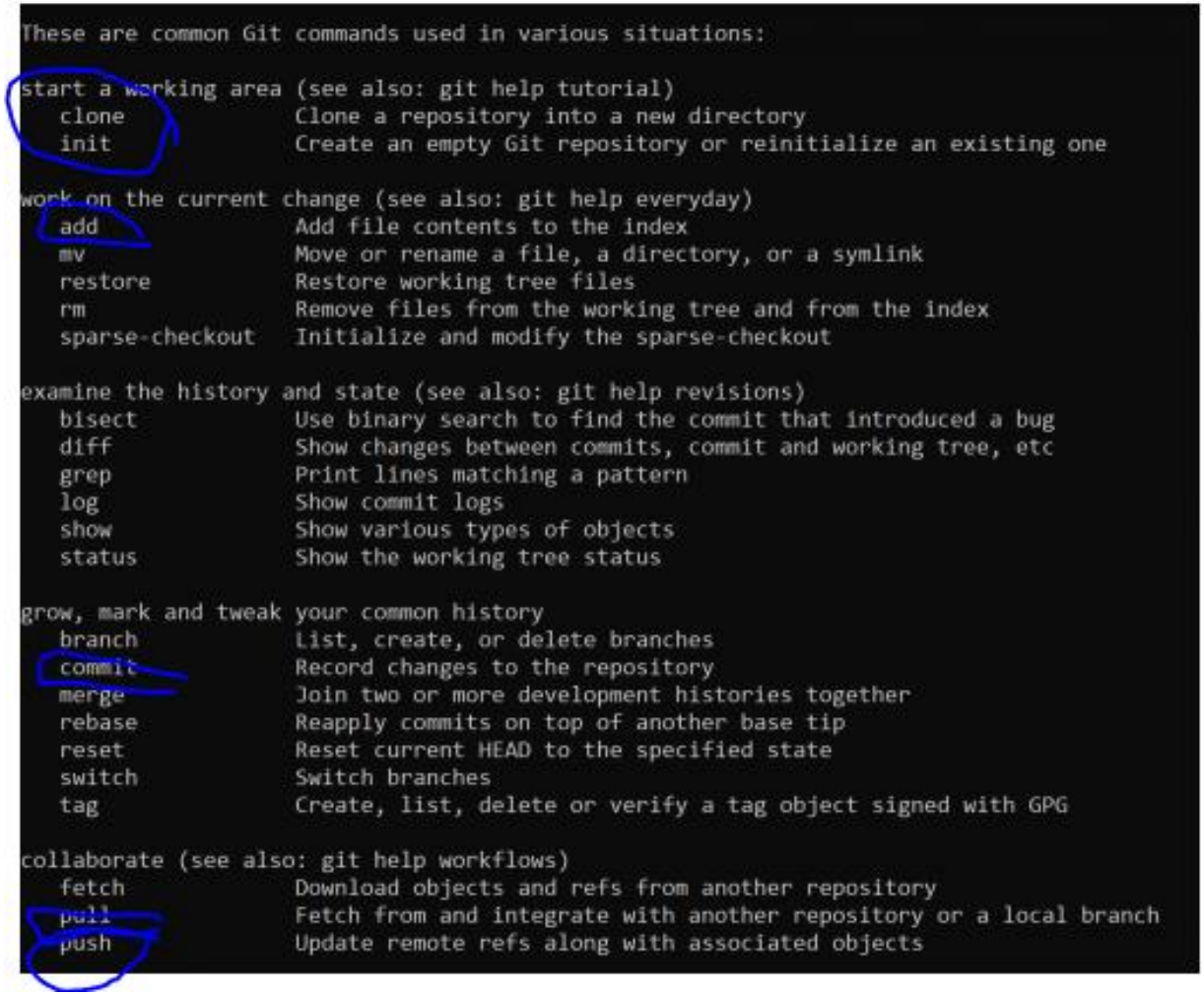
DevOps Tools : Git

- **Create an account and login to github.com**
  - You can get the git url by clicking on Code



DevOps Tools : Git

- **Commonly Used git commands**



```
These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
   clone             Clone a repository into a new directory
   init              Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
   add               Add file contents to the index
   mv                Move or rename a file, a directory, or a symlink
   restore           Restore working tree files
   rm                Remove files from the working tree and from the index
   sparse-checkout   Initialize and modify the sparse-checkout

examine the history and state (see also: git help revisions)
   bisect            Use binary search to find the commit that introduced a bug
   diff              Show changes between commits, commit and working tree, etc
   grep              Print lines matching a pattern
   log               Show commit logs
   show              Show various types of objects
   status            Show the working tree status

grow, mark and tweak your common history
   branch            List, create, or delete branches
   commit            Record changes to the repository
   merge             Join two or more development histories together
   rebase            Reapply commits on top of another base tip
   reset             Reset current HEAD to the specified state
   switch            Switch branches
   tag               Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)
   fetch             Download objects and refs from another repository
   pull              Fetch from and integrate with another repository or a local branch
   push              Update remote refs along with associated objects
```

DevOps Tools: Jenkins

- **Jenkins is an open-source build automation tool to build, test, and deploy software.**
- **Can be used to easily set up continuous integration and continuous delivery (CI/CD) pipelines.**
  - Continuous integration (CI) is a DevOps practice in which team members regularly commit their code changes to the version control repository, after which automated builds and tests are run.
  - Continuous delivery (CD) is a series of practices where code changes are automatically built, tested, and deployed to production.
  - Jenkins can be installed as a standalone application, or can be run as a Docker container.

DevOps Tools: Jenkins

- **Let's install Jenkins on Ubuntu 20.04 as a standalone service.**
- **Jenkins is a Java application and requires Java 8 or later to be installed on the system.**
  - You can install OpenJDK 11 , the open-source implementation of the Java Platform, or any late version.

```
$ sudo apt update
$ sudo apt install openjdk-11-jdk
```

```
$ java -version
```

```
Output
openjdk version "11.0.7" 2020-04-14
OpenJDK Runtime Environment (build 11.0.7+10-post-Ubuntu-3ubuntu1)
OpenJDK 64-Bit Server VM (build 11.0.7+10-post-Ubuntu-3ubuntu1, mixed mode, sharing)
```

DevOps Tools: Jenkins

- **Installing Jenkins on Ubuntu involves:**
  - Enable the Jenkins APT repository, import the repository GPG key, and
  - Install the Jenkins package.

- **Import the GPG keys of the Jenkins repository using the following wget command:**
  - $ wget -q -O - https://pkg.jenkins.io/debian/jenkins.io.key | sudo apt-key add –

- **Next, add the Jenkins repository to the system with:**
  - $ sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'

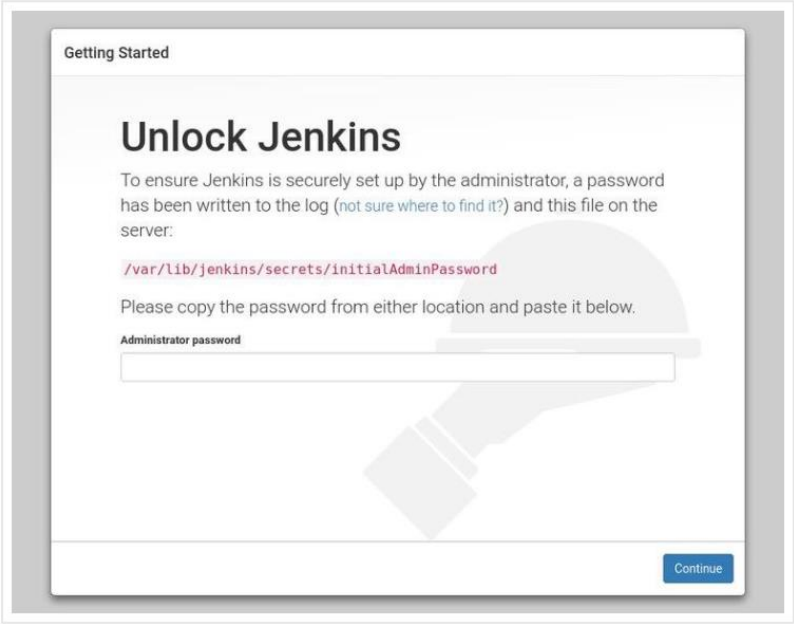
- **Once the Jenkins repository is enabled, update the apt package list and install the latest version of Jenkins by typing:**
  - $ sudo apt update
  - $ sudo apt install Jenkins

  - Jenkins service will automatically start after the installation process is complete. You can verify it by printing the service status:
    - $ systemctl status jenkins
    - **Output:**

      jenkins.service - LSB: Start Jenkins at boot time
      Loaded: loaded (/etc/init.d/jenkins; generated)
      Active: active (exited) since Thu 2020-07-16 20:22:12 UTC; 15min ago
      ...

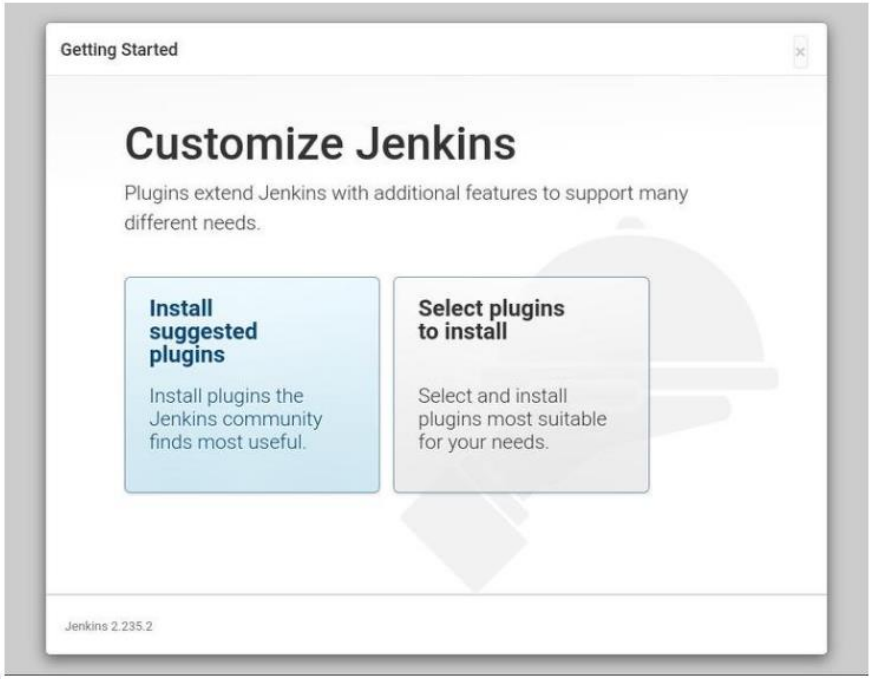
DevOps Tools: Jenkins

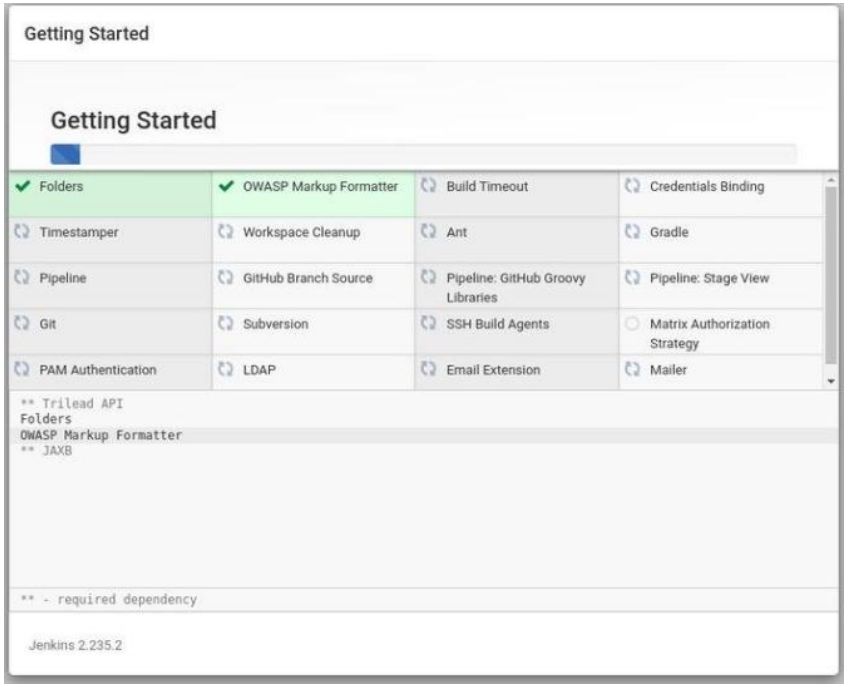
- **Setting Up Jenkins**
  - To set up your new Jenkins installation, open Jenkins getting started page using http://your_ip_or_domain:8080.
    - A page similar to the following will be displayed, prompting you to enter the Administrator password that is created during the installation:
  - Get the password on the terminal using cat command:
    - $ sudo cat /var/lib/jenkins/secrets/initialAdminPassword
    - You should see a 32-character long alphanumeric password, as shown below:
    - 06cbf25d811a424bb236c76fd6e04c47
  - Copy the password from the terminal, paste it into the "Administrator password" field and click "Continue".



DevOps Tools: Jenkins

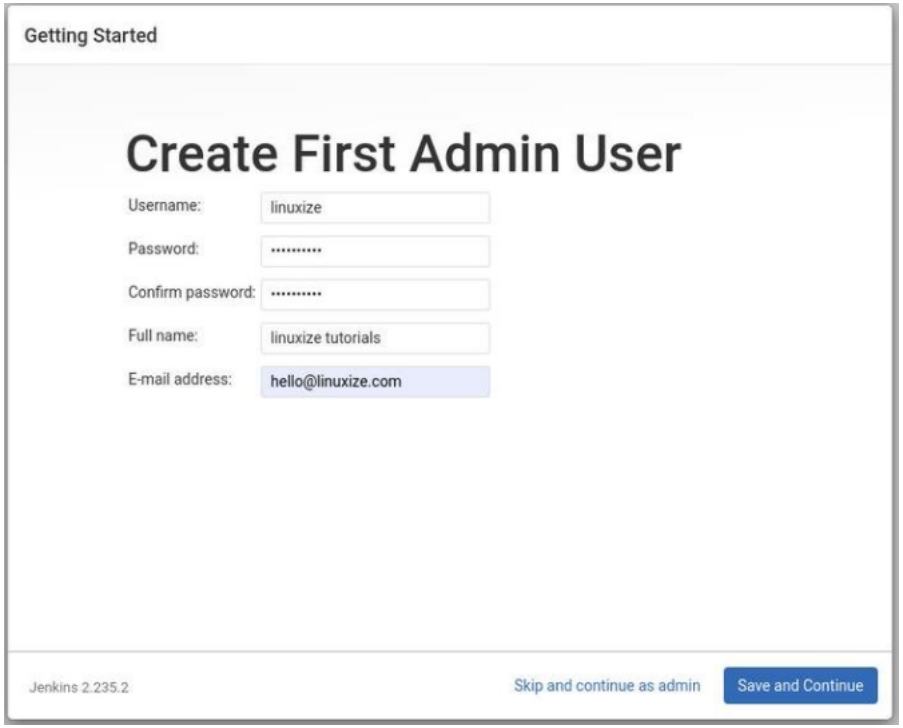
- **Setting Up Jenkins**
  - On the next page, click on the "Install suggested plugins" box and the installation process will start immediately.





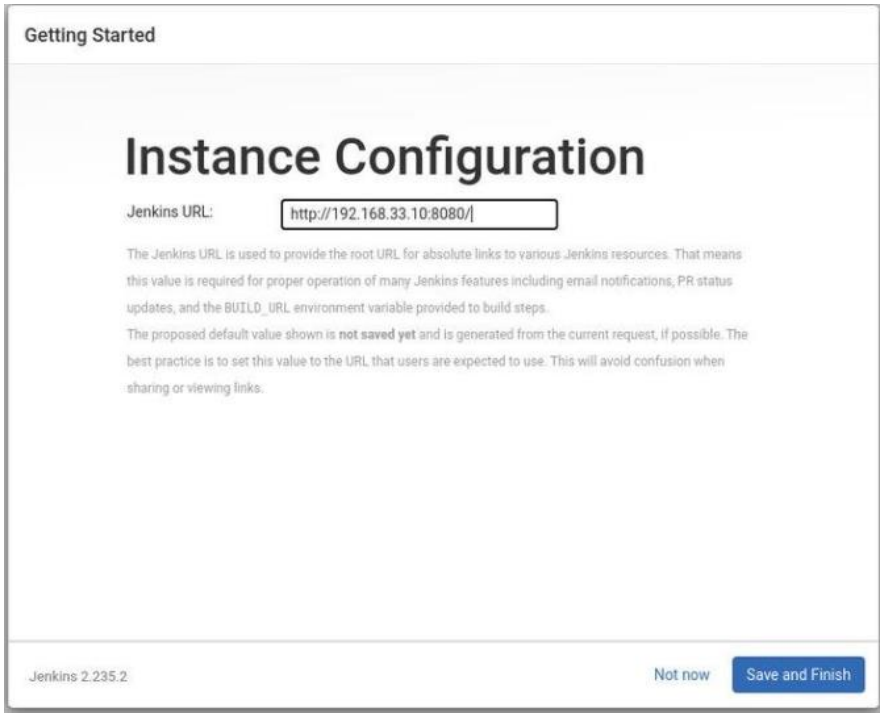
DevOps Tools: Jenkins

- **Setting Up Jenkins**
  - Once the plugins are installed, you will be prompted to set up the first admin user.
  - Fill out all required information and click “Save and Continue”.



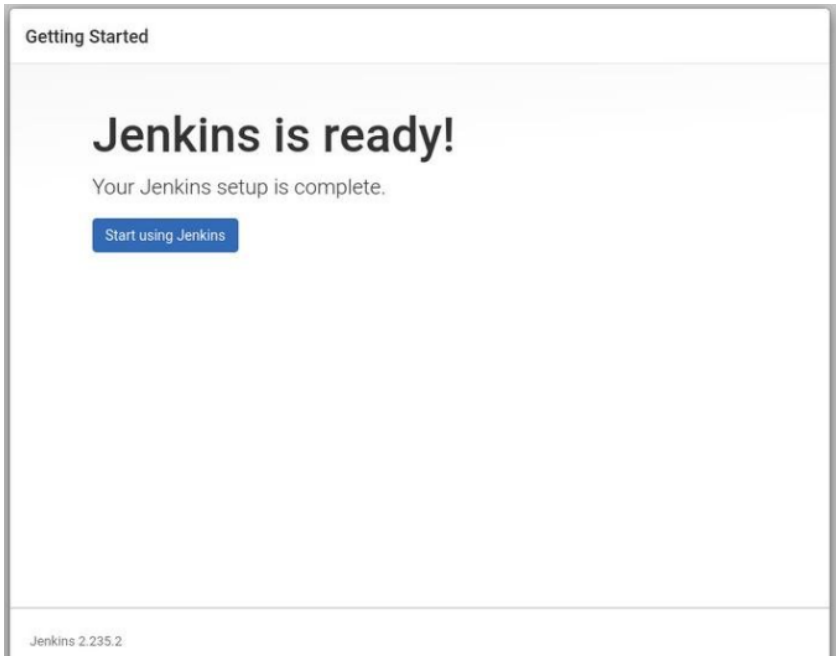
DevOps Tools: Jenkins

- **Setting Up Jenkins**
  - The next page will ask you to set the URL for your Jenkins instance.
  - The field will be populated with an automatically generated URL.



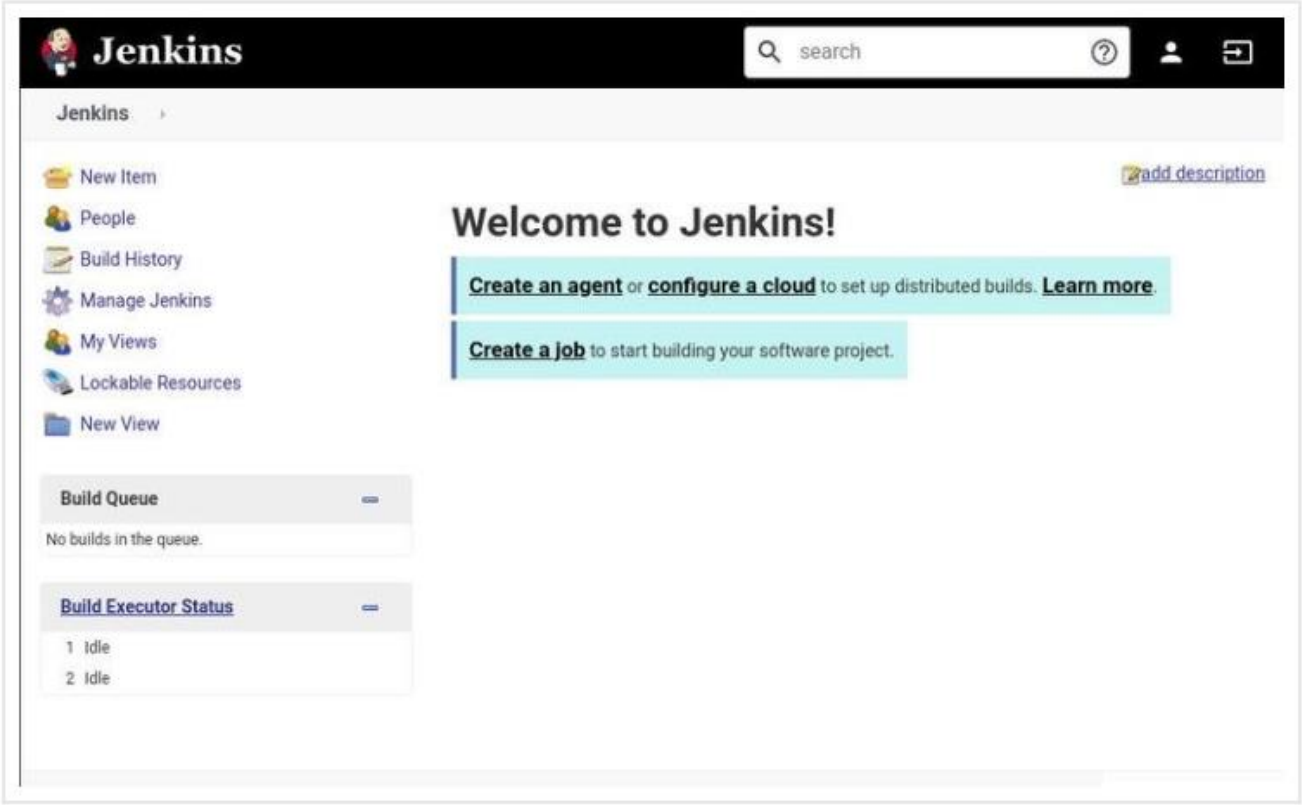
DevOps Tools: Jenkins

- **Setting Up Jenkins**
  - Confirm the URL by clicking on the Save and Finish button, and the setup process will be completed.
  - Click on the Start using Jenkins button, and you will be redirected to the Jenkins dashboard logged in as the admin user you have created in one of the previous steps.



DevOps Tools: Jenkins

- **Setting Up Jenkins**
  - At this point, you've successfully installed Jenkins on your server.

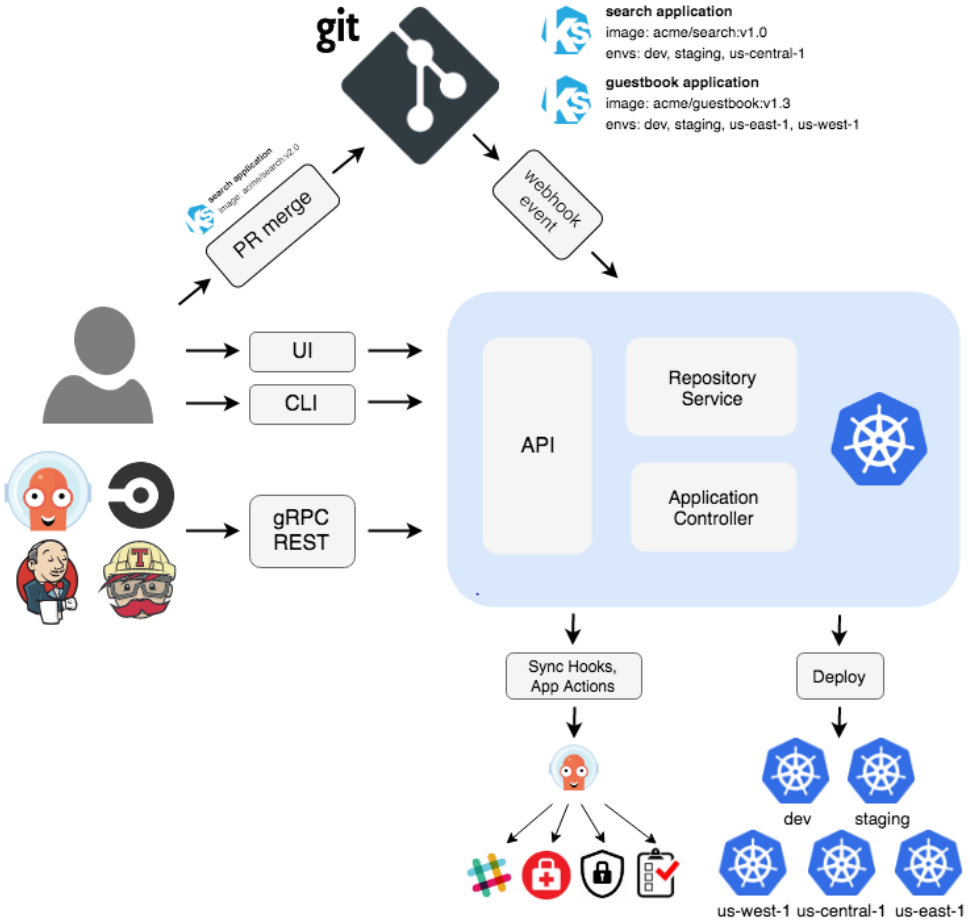


DevOps Tools: Jenkins

- **Jenkins Job**
  - Create a job and choose a github project.
  - Input the URL of the git repository.
  - Set the trigger to poll scm every minute
  - Perform operations in the build shell

Argo CD

- Argo CD is a declarative continuous deployment tool to deploy apps on Kubernetes cluster.
  - You define all your application manifests, (e.g., YAMLs) that you want to deploy on Kubernetes cluster in a github repository.
  - Argo CD will pull the resource definitions or all changes from github repository and deploy those resources for you on your Kubernetes cluster.



Source: https://argoproj.github.io/argo-cd/

Argo CD

- Argo CD is comprised of the following key components:
  - API server – the Argo CD Server –all interaction with Argo CD goes through this API server. It's like the Kube API server for Kubernetes.
  - Repository Service - responsible for communicating with external github repository, and then maintains a cache of git hub repository.
    - Whenever you make any changes to the github repository, it polls every now and then to see what has changed and what's the current state of the deployment.
  - Application Controller - responsible for maintaining the state of deployed resources in your Kubernetes cluster
  - Other components include: Redis - for caching and Dex server

Deploying through Argo CD

- For example, let's say you want to deploy a web application, so need to create a Deployment – you may also setup Ingress to expose your application as a service
  - Deployment creates ReplicaSet, and the ReplicaSets manages the pods. (more on this later)
- ArgoCD manages this Deployment as one single application
- Steps involve:
  - Define YAML manifest for your deployment, service, and other resources that you want to deploy on your Kubernetes cluster in a github repository
    - that holds manifests for your applications
  - Once Argo CD is deployed in your Kubernetes cluster, you need to create a repository in Argo CD

Deploying through Argo CD

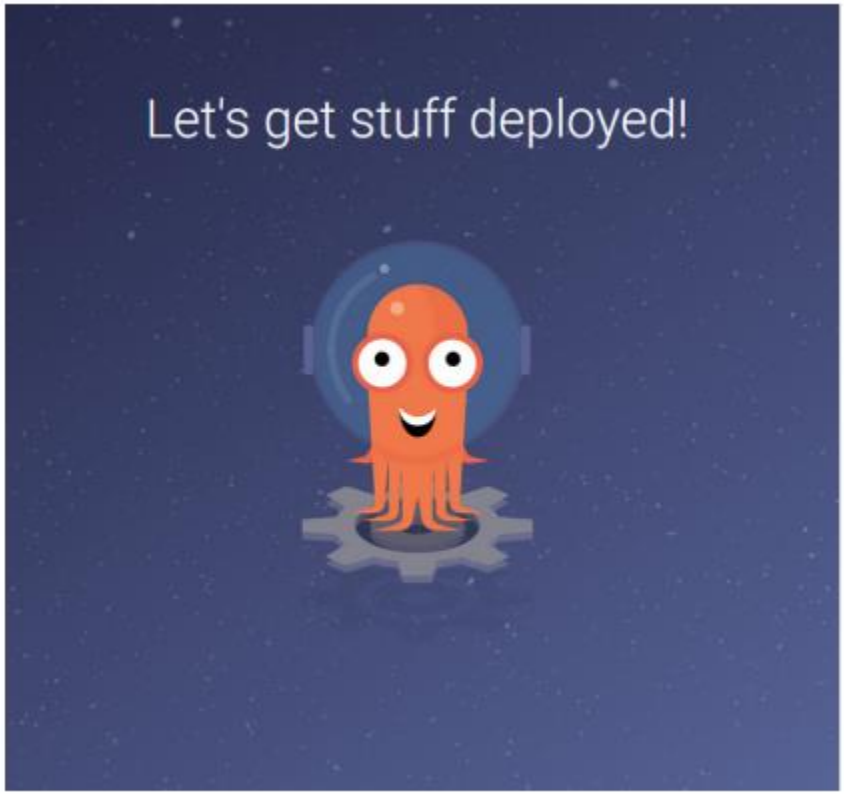
- To interact with ArgoCD you can use its Web UI to create and manage applications and sync your resources.
- You can also use the Argo CD command line interface.
- More on Argo CD later
  - How to install/Deploy Argo CD on Kubernetes cluster using helm charts
  - How to deploy application manifests to deploy on Kubernetes
  - How to use Argo CD CLI

Deploying through Argo CD

- Installing Argo CD – on a Kubernetes cluster
  - kubectl create namespace argocd
  - kubectl apply -n argocd –f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml
- # Change the service type to load balancer so it can be accessed
  - kubectl patch svc argocd-server -n argocd -p '{"spec": {"type": "LoadBalancer"}}'

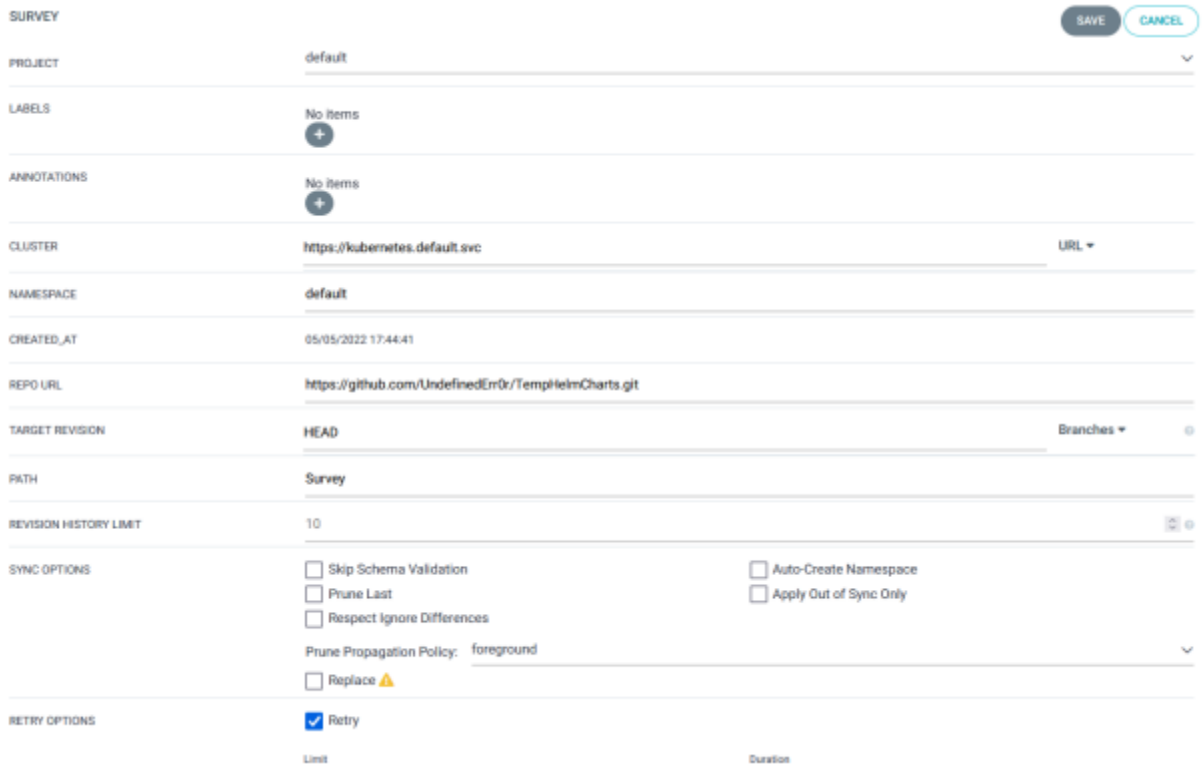
Deploying through Argo CD

- Access the ArgoCD Interface on the kubernetes endpoint.
- You are greeted with a login screen.
- The login details can be viewed by the following command.
  - kubectl -n argocd get secret argocd-initial-admin-secret -o jsonpath="{.data.password}" | base64 -d; echo



Deploying through Argo CD

- **Setup a New ArgoCD Job**
  - Input the name of the deployment Repo. The deployment repo just consists of a folder “Survey” with one yaml file.
  - The folder name needs to be entered under PATH.
  - Scroll down and select auto-sync.



Deploying through Argo CD

- Once the sync starts, you should see the following status:
- You have option to manually synchronize as well

Benefits of DevOps

- **Improved Communication and Collaboration**
  - Increased Efficiencies
- **Shorter Development Cycles**
  - Faster Innovation
- **Reduced Deployment Failures**
  - Reduced Rollbacks, and Time to Recover
- **Reduced Costs and IT Headcount**

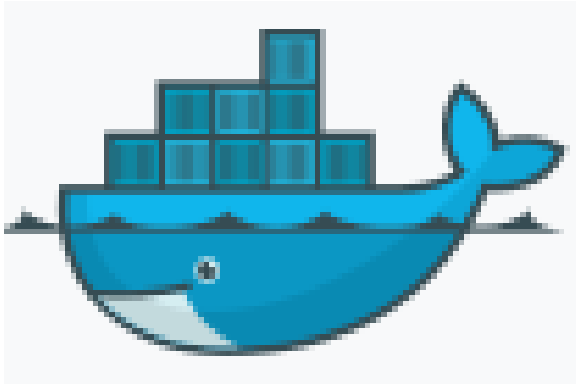
Containers

Problem

- **Deploying the Code**
  - How to build an efficient, lightweight, self-contained systems?
  - How to guarantee to run the same way, regardless of where it's deployed?
  - How to address application dependencies?
  - How to abstract the infrastructure and OS?
    - Allowing you to focus on developing business capabilities
  - How to achieve higher performance?
    - Without the overhead of Hypervisor

Problem

- **Deploying the Code**
  - How to build an efficient, lightweight, self-contained systems?
  - How to guarantee to run the same way, regardless of where it's deployed?
  - How to address application dependencies?
  - How to abstract the infrastructure and OS?
    - Allowing you to focus on developing business capabilities
  - How to achieve higher performance?
    - Without the overhead of Hypervisor
- **Solution:**
  - Containers
    - Build containerized applications
    - Docker containers



What is a Container?

- Containers are a way to package software (e.g., your application) with all its dependencies in a format that can run isolated on a shared operating system.
- Enables flexibility and portability on where the application can run, exactly the same
  - whether on premises, cloud, VMs, laptops, or bare metal
- Provides consistency in application deployments
- Analogy –mobile phone – you download a self contained application and run it
  - Containers do the same thing for your applications

What is a Docker?

- **A tool to containerize your application**
  - Enables packaging of an application and all its dependencies in a virtual container
    - It puts everything that your application needs into an image that can be run on any computer that has Docker engine/runtime on it.
- **Docker provides a nice API**
  - that makes it easy to create, and run container images on your servers
  - The de-facto standard for container format and runtime
- **Docker Engine comes with Docker installation and is the one that creates and runs docker containers**

Source: https://www.linux.com/news/docker-shipping-container-linux-code

Installing Docker

- **Installing docker on Mac**
  - Install Docker onto your machine from Docker Desktop for Mac or Window.
  - You can install drocker from Docker Desktop on Mac from
    - https://docs.docker.com/desktop/mac/install/
  - Once dmg is downloaded, drag it into the Applications folder.
  - On the command line, type ‘docker -v’ and you should see the output similar to the following:
    - Docker version 20.10.8, build 3967b7d

Installing Docker

- **Installing Docker EC2 (Ubuntu)**
  - Launch a new Ubuntu EC2 instance
  - Run updates on your machine using the command
    - $ sudo apt-get update
  - Install docker on your machine using the command
    - $ sudo apt install docker.io
  - Verify docker is installed on your machine using the command
    - $ sudo docker -v
  - To avoid using sudo in front of every docker command, grant your ubuntu user permissions to perform docker commands by running the command
    - $ sudo usermod -aG docker ubuntu
  - Log out and log back in for the changes to take place.
    - You can run docker command without using sudo

What is a Docker Image?

- **A Docker Image is an immutable file that's essentially a snapshot of a container.**
  - Analogy: AMI for EC2 instance
- **A package format that includes**
  - Your application and all its dependencies and runtime information required to run it
- **Docker images are created with the docker build command**

What is a Docker Image?

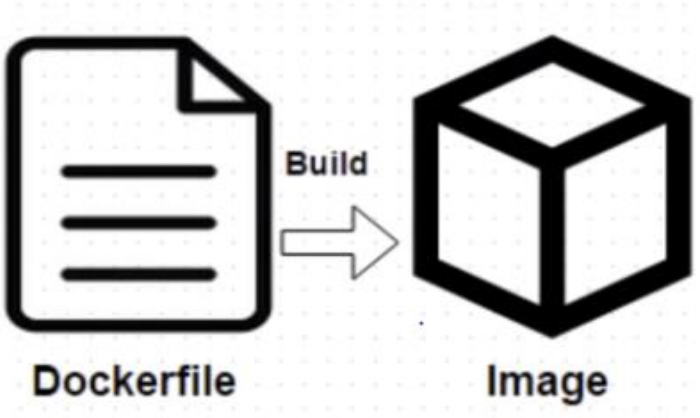
- **A docker images produces a container when started with docker run command**

- **Essentially, containers are running instances of Docker images**
  - An analogy with Java class/objects:
    - Docker Image as Java class and
    - Docker container as the object or an instance of the class

Dockerfile – How do you create an image?

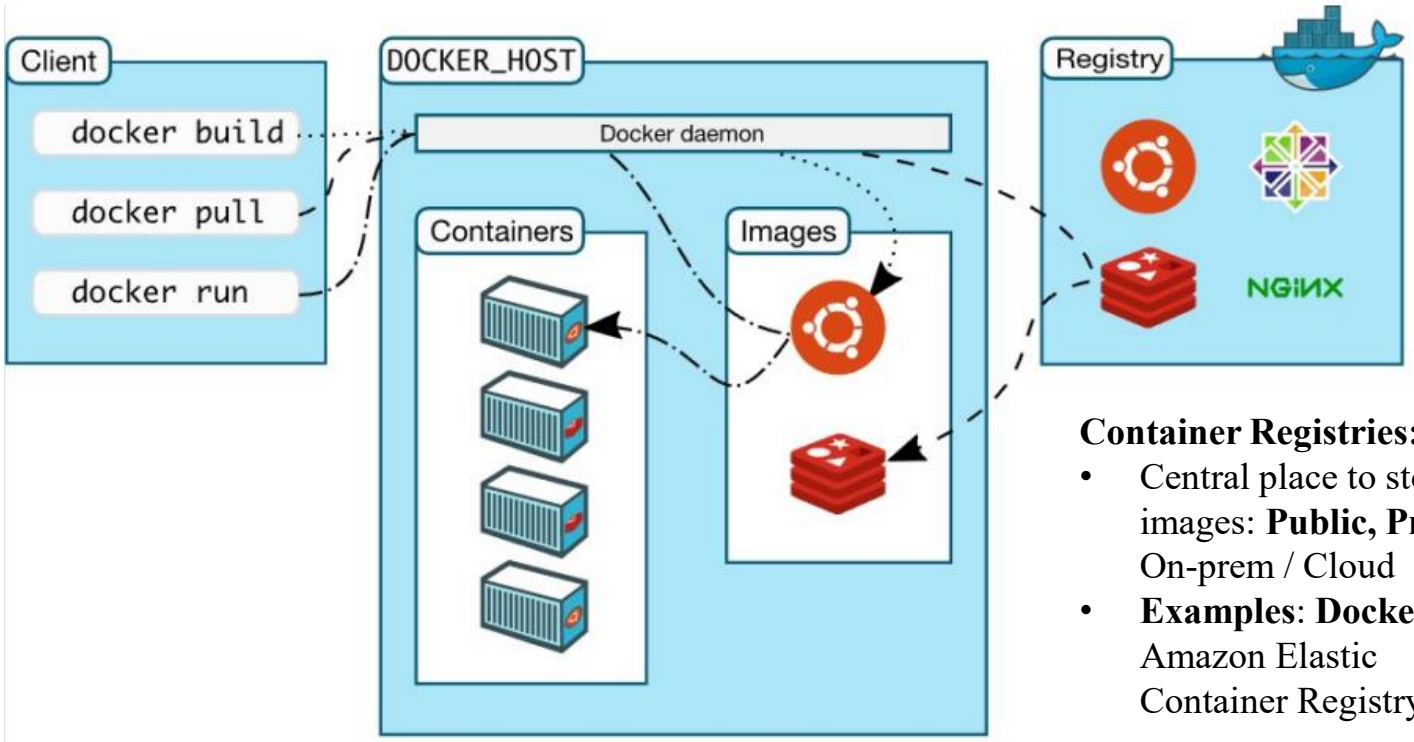
- **To create an image, you define a Dockerfile**
- **Dockerfile** is a text document that **contains** all the **commands** a user could call on the command line to assemble an image.
  - Using docker build users can create an automated build that executes several command-line instructions in succession.
  - Once you have written your Dockerfile, you can then build your program from the Dockerfile into an image and that image can then be run as a container on any computer that has Docker.

- Describes the build process for an image
- Can be run to automatically create an image
- Contains all the commands necessary to build the image and run your application



Docker Architecture

- **Docker Architecture**

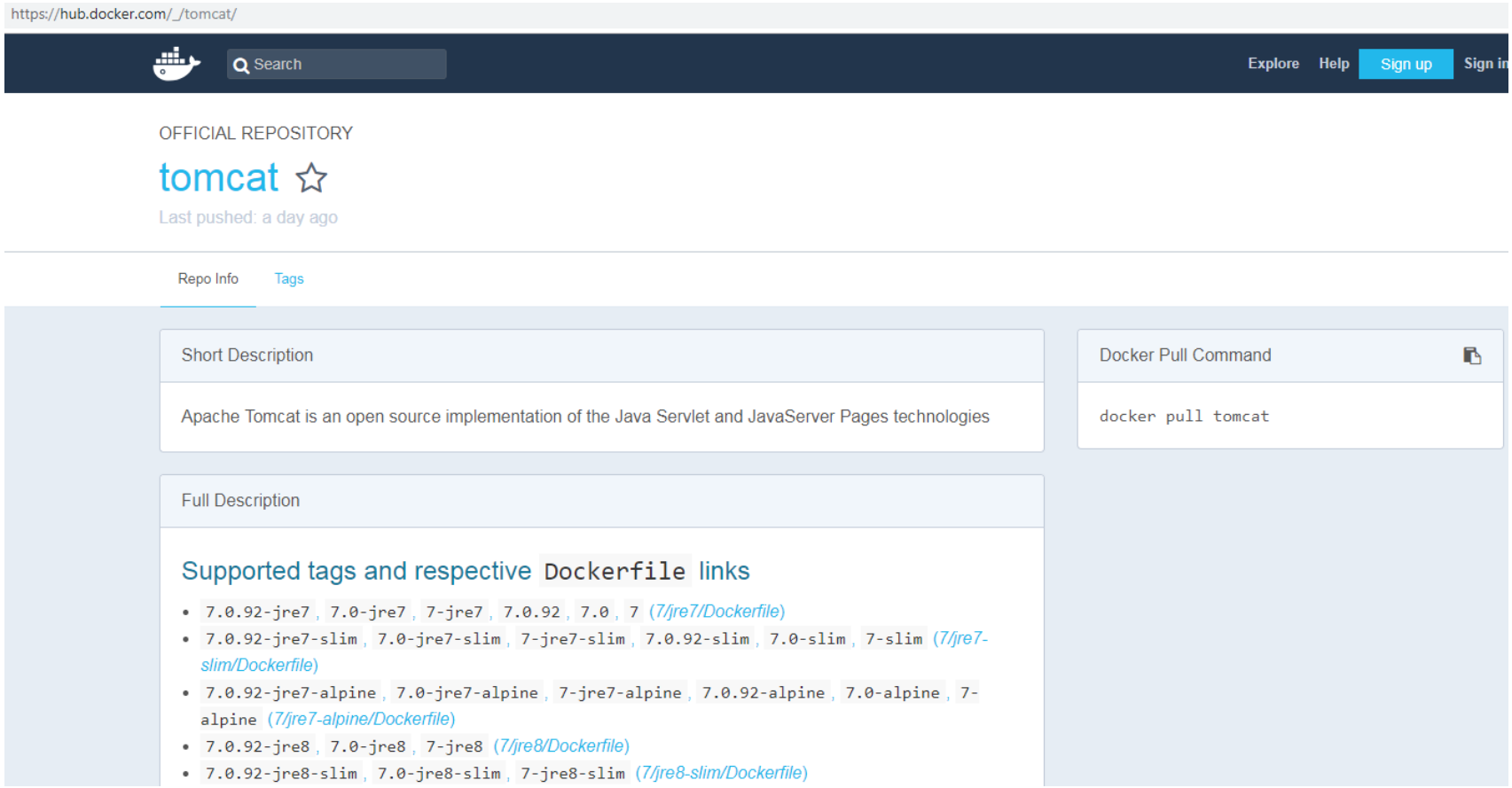


Container Registries:

- Central place to store images: **Public, Private**, On-prem / Cloud
- **Examples**: **Docker Hub**, Amazon Elastic Container Registry (ECR)

Tomcat base image

- **URL to access tomcat base image**



Packaging and running Containers

- **Dockerfile** to build the container that runs myapp

```
FROM tomcat:8.0.20-jre8

RUN mkdir /usr/local/tomcat/webapps/

COPY /1.0-SNAPSHOT/my-app-1.0-SNAPSHOT.war /usr/local/tomcat/webapps/
```

- To start the container from the created image.

```
docker build -t myapp .

docker run -it --rm -p 8888:8080 myapp
```

22-Mar-2015 23:07:21.217 INFO [localhost-startStop-1] org.apache.catalina.startup.HostConfig.deployDirectory Deploying web application directory /usr/local/tomcat/webapps/myapp

- Container starts and displays the message that myapp was deployed.
- The container copies my-app-1.0-SNAPSHOT.war to the path described in the Dockerfile.
- You can navigate to Tomcat default page to confirm that Tomcat is running
- Or you can access your website by typing the following in a browser:
  - http://localhost:8888/[WarFileName]/[html file]

Running Docker on AWS EC2

- **Setup an EC2 instance**
  - Using Amazon Linux AMI
  - Configure the **security groups** allowing access to port **80** (**HTTP**) from anywhere, and **SSH** access also
- **Install Docker on EC2 instance**
  - SSH to the EC2 instance using the public DNS and the public key
  - ***sudo yum update -y***
  - ***sudo yum install -y docker***
- **To start the docker service:**
  - ***sudo service docker start***
- **To see if the docker got installed, type:**
  - ***sudo docker info***
- **To list running instances if any**
  - ***sudo docker ps***
- **To use docker command without root privileges (sudo), add ec2-user to the docker group:**
  - ***sudo usermod -aG docker ec2-user***

Running Docker on AWS EC2

- **Deploy Docker Container (e.g., nginx)**
  - docker run –d –p 80:80 nginx nginx

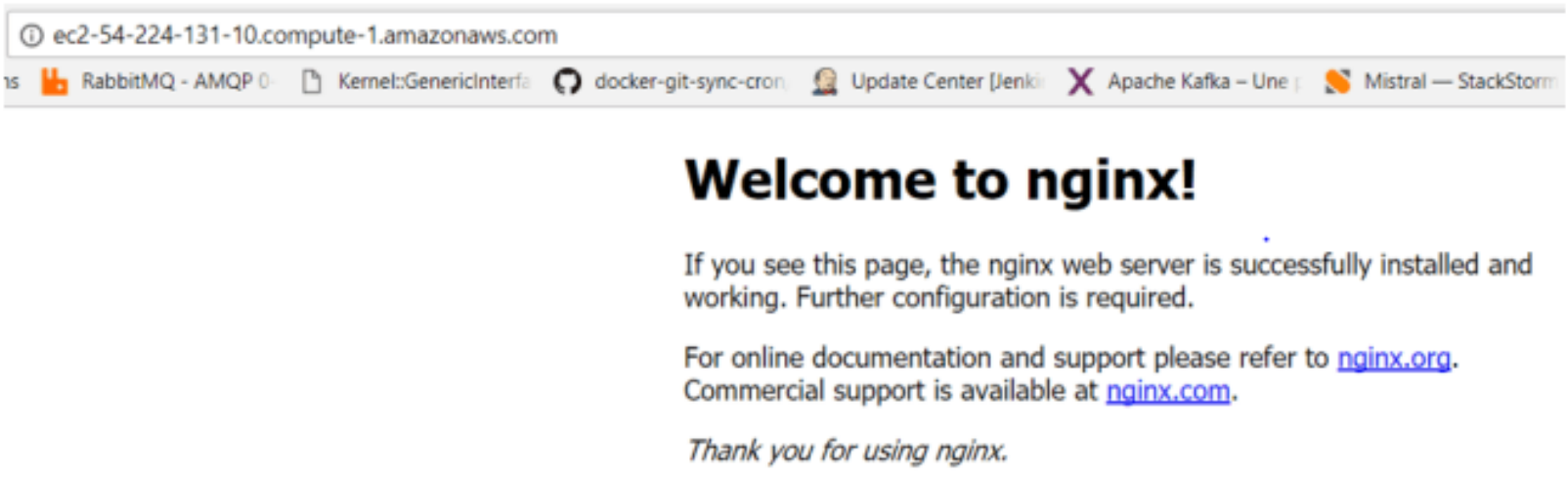
```
[ec2-user@ip-172-31-8-51 ~]$ docker run -d -p 80:80 --name nginx nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
94ed0c431eb5: Pull complete
9406c100a1c3: Pull complete
aa74daafd50c: Pull complete
Digest: sha256:788fa27763db6d69ad3444e8ba72f947df9e7e163bad7c1f5614f8fd27a311c3
Status: Downloaded newer image for nginx:latest
b60fe57f39f49b7de72e6ceff7d1333ea5b2f6a13952064a831cd6345e8b5c3c
[ec2-user@ip-172-31-8-51 ~]$
```

- **Run “docker ps” again to see that an nginx container has been created from the nginx official image.**

```
ec2-user@ip-172-31-8-51 ~]$ docker ps
ONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                  NAMES
60fe57f39f4        nginx               "nginx -g 'daemon ..."   12 seconds ago      Up 12 seconds       0.0.0.0:80->80/tcp     nginx
ec2-user@ip-172-31-8-51 ~]$
```

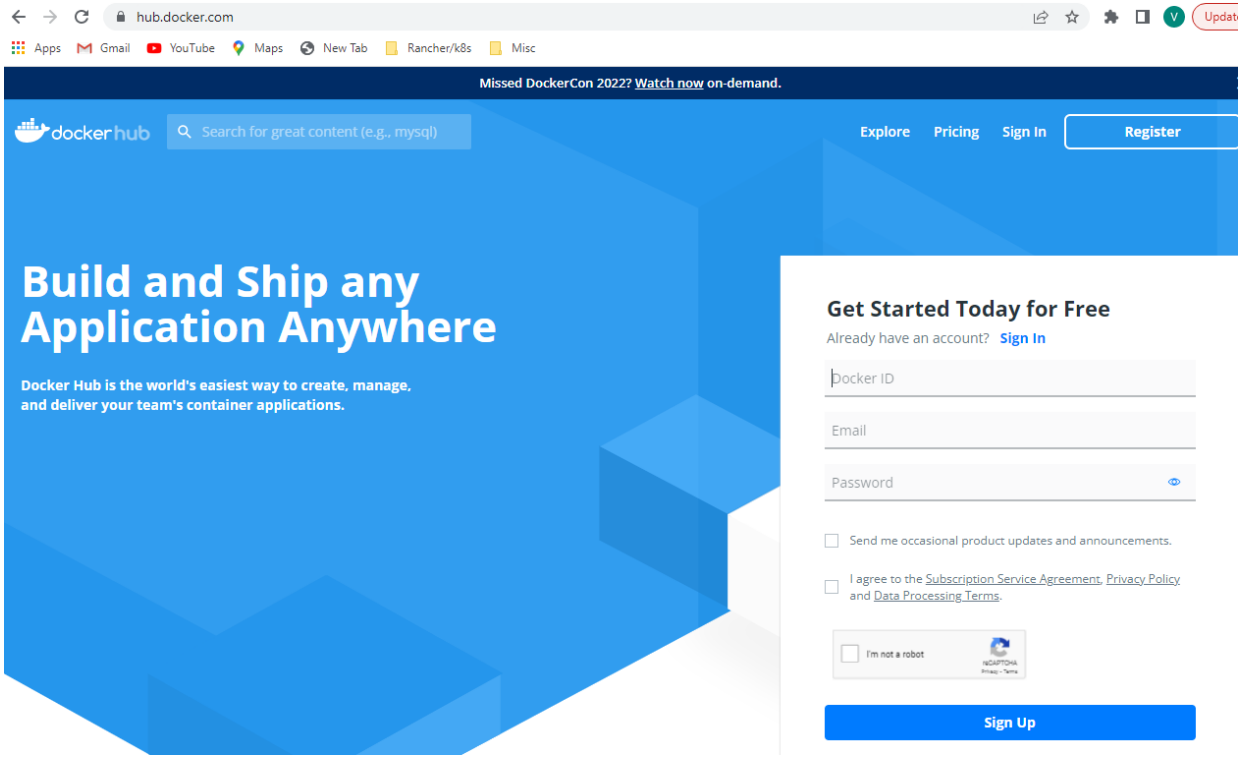
Running Docker on AWS EC2

- **Visit your instance public DNS name in your browser to see something like this below:**



Upload image to DockerHub

- **To be able to deploy your custom docker images on Kubernetes cluster, you will need to push the docker image to docker hub**
  - You will need to create an account on hub.docker .com

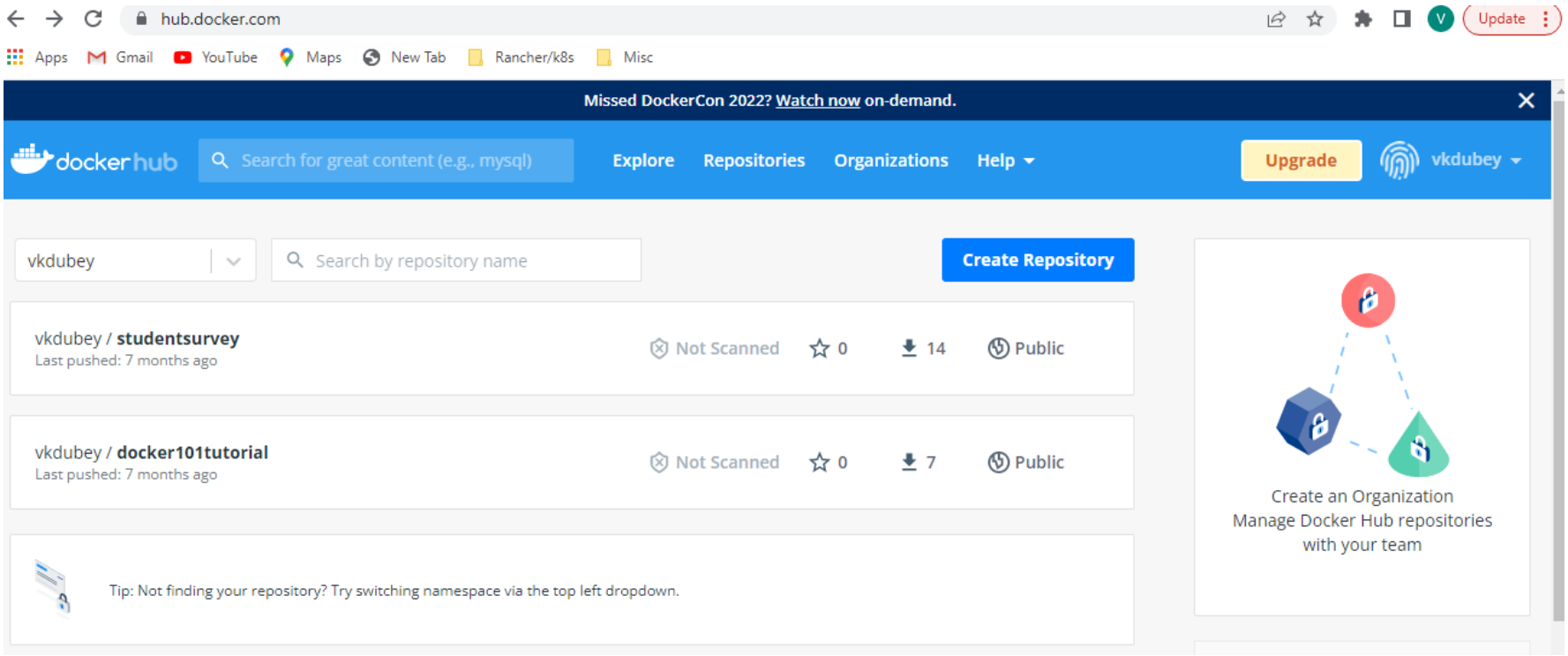


Upload image to DockerHub

- **To push the docker image to docker hub**
  - On the command line, login to docker using ‘
    - $ docker login -u <your username>
  - Change the name of your image to be <your username on dockerhub>/<name of the app>:<image tag> using the docker tag command. For example:
    - $ docker tag vkdubey/surveyapi vkdubey/studentsurvey:1.0 .
  - Use command to push the image on to your docker hub
    - $ docker push vkdubey/studentsurvey:1.0
  - Verify that your image is on Docker Hub.
    - Your image is accessible from the internet.

Docker Hub

- **Hub.docker.com**



Container Orchestration

Issue

- **Starting and stopping one container in Dev on your laptop is easy!**
- **Managing a cluster of applications in production is a hard problem**
  - How to make containers resilient?
  - How to achieve horizontal scalability across multiple servers?
  - How to group related containers so that they run on the same host in order to work
  - How to roll out new version of my software without any service interruption?

Issue

- **Starting and stopping one container in Dev on your laptop is easy!**
- **Managing a cluster of applications in production is a hard problem**
  - How to make containers resilient?
  - How to achieve horizontal scalability across multiple servers?
  - How to group related containers so that they run on the same host in order to work
  - How to roll out new version of my software without any service interruption?
- **Answer: Container Orchestration**

What is Container Orchestration?

- **A general term for technologies that enable managing large collection of containers easy**
  - Allows deploying docker containers on a cluster and scale
  - Automates container lifecycle
  - The automated configuration, coordination, and management of containers
- **Key Responsibilities**
  - Provisioning and deployment of containers
  - Redundancy and availability of containers
    - Resiliency and fault tolerant
  - Elasticity and load balancing
  - Movement of containers between hosts
  - Container access control (ingress/egress)
  - Allocation of container resources
  - Health and monitoring of containers and hosts

What is Container Orchestration?

- **Popular container orchestrators**
  - Kubernetes (**K8s**) by Google
  - Docker Swarm
    - The official orchestration platform by Docker
  - Google Kubernetes Engine (GKE)
    - runs Kubernetes under the hood
  - EKS by AWS
  - Azure Kubernetes Service (AKS)
  - Mesosphere DC/OS
  - Marathon

What is Kubernetes?

- **Kubernetes is an open-source platform for managing containerized workloads and services**
- **An open-source container-orchestration system**
  - for automating deployment, scaling, and management of containerized applications
- **It was originally designed by Google**
  - Supported by vibrant and growing community of users and contributors
  - Kubernetes can run anywhere – on premises or cloud!

What is Kubernetes?

- **Kubernetes takes declared state of how you would like to configure your containers in a form of YAML file and it makes that happen, even across computers.**
  - It will deploy your containers and make them publicly available, among other things.

Kubernetes Vocabulary

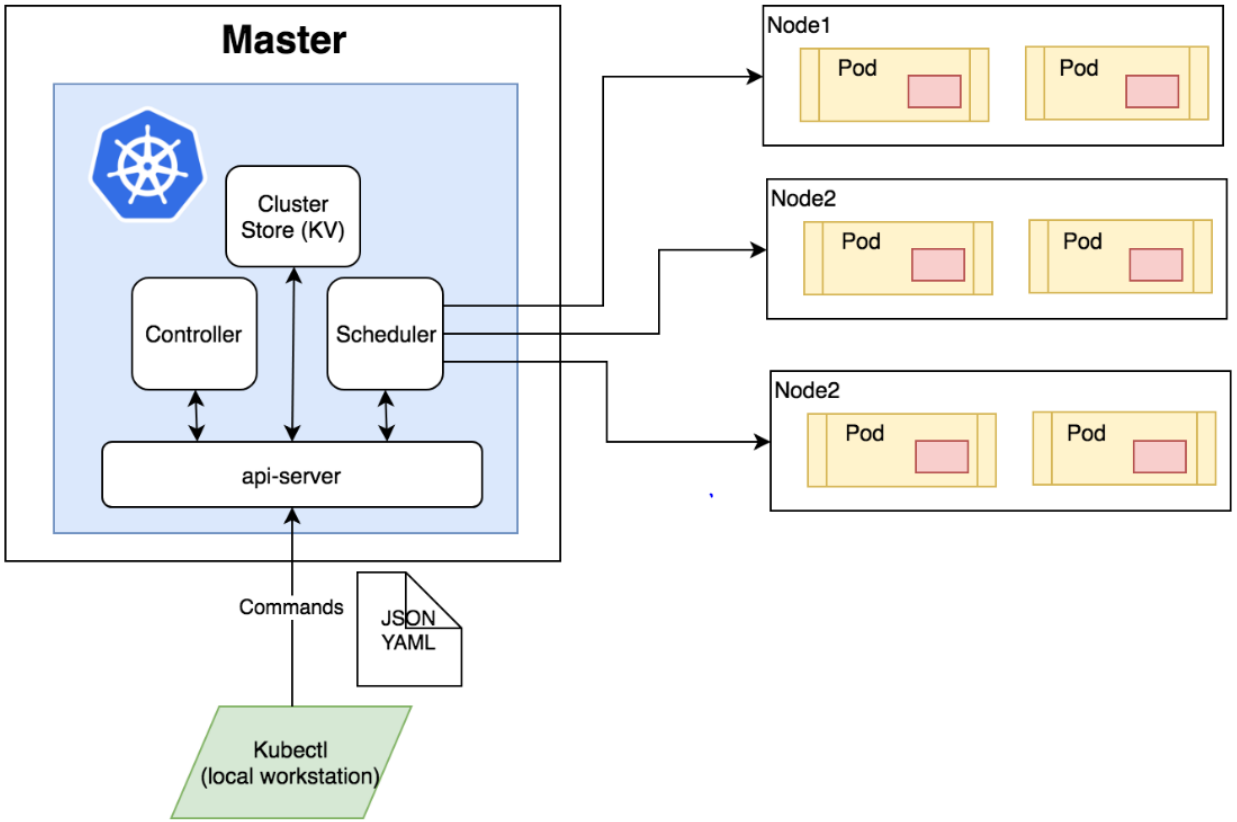
- **Cluster**
  - A collection of physical or virtual machines working together
- **Node**
  - A physical or virtual machine running Kubernetes master or Worker processes and able to schedule pods
    - Kubelet is the application that runs on worker nodes and that communicates with the master node.
    - A node runs pods.
- **Pod**
  - A pod is a smallest unit of deployment
    - There can be one container or many containers running on a pod – generally one container per pod
    - A group of containers that share a common environment (volumes, network, shared memory, etc.)
    - Pods run (or exist) on a Node.

Kubernetes Vocabulary

- **Deployment**
  - Deployment defines desired state, and Kubernetes makes that happen for you
  - An entity that describes a pod's containers and replicas and allows for multiple copies of a replica per cluster node
- **Service**
  - A reliable networking endpoint for a set of pods
  - A service handles requests,
    - either coming from inside the Kubernetes cluster, from one node to another or from outside the cluster, say public request to our master node that wants to hit a specific microservice.
  - A service tends to be a Load Balancer
    - there are a couple different types that you can declare, but services are essentially definition of how requests should be routed and handled within the cluster.
- **Ingress**
  - Kubernetes ingress is a collection of routing rules that govern how external users access services running in a Kubernetes cluster.

Kubernetes Cluster

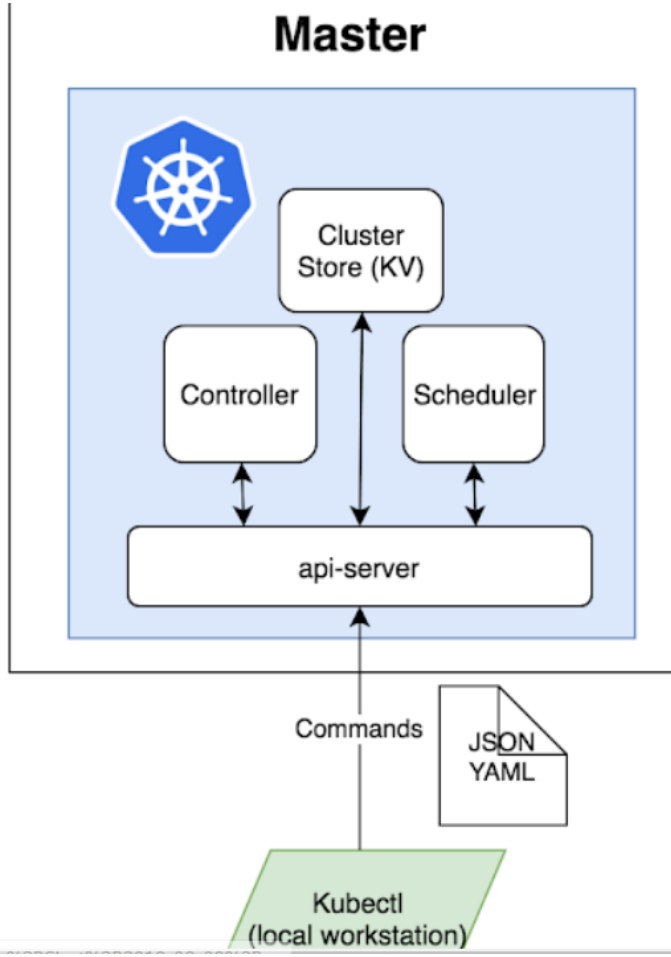
- **This is a visualization of what a Kubernetes cluster would look like**



Source: Google

Kubernetes Master

- **K8S Master is responsible for managing the entire cluster**
  - It coordinates all activities inside the cluster and communicates with the worker nodes to keep K8S and your applications running
- **When you install Kubernetes on your system, there are 4 primary components of the Kubernetes master that get installed**
  - API Server
  - Schedular
  - Control manager
  - Cluster Store - etcd



Kubernetes Master

- **API Server**
  - The API server serves as a gatekeeper for the entire cluster.
    - If you want to create, delete, update or even display any Kubernetes object, it has to go through that API.
  - It is responsible for exposing various APIs.
    - It exposes APIs for almost every operations
  - Exposes REST API to talk to K8s cluster
    - Only api-server talks to Cluster Store.
- **We interact with API server using a tool called kubectl**
  - which is a tiny go language binary.
  - It basically talks to API server to perform any operations that we issue from command line

Kubernetes Master

- **Schedular**
  - Schedular is responsible for physically scheduling pods across multiple nodes.
  - Watches api-server for new pods and assign node to work
  - Depending on the constraints that you mentioned in the configuration file, the schedular schedules these pods accordingly
    - For example, if you mention CPU 1 core, memory 10 Gig, disk type is SSD, and other affinity or constraints that you may want to declare in the artifact.
    - So, once you pass these artifacts to the API server, the schedular will look for the appropriate nodes that meets the specified criteria and will schedule the pods accordingly

Kubernetes Master

- **Control Manager or Controller**
  - A daemon that watches the state of the cluster to maintain desired state.
    - Example are replication-controller, namespace-controller etc. Other than this it performs garbage collection of pods, nodes, events etc.
  - There are 4 controllers behind the control manager:
    - Node controller,
    - Replication controller,
    - End point controller, and
    - Service accountant token controller
  - At a high level, these controllers are responsible for the overall health of the entire cluster
    - It ensures that nodes are up and running all the time; and also that the correct number of pods are running as mentioned in the spec file

Kubernetes Master

- **etcd**
  - Etcd is a distributed key-value lightweight database
  - Etcd in Kubernetes is the central database, a key-value database to store the current cluster state at any point of time.
  - Any component of Kubernetes can query etcd to understand the current state of the cluster
  - This is going to be the single source of truth for all the nodes, all the components, and the master that are forming Kubernetes cluster
  - **Cluster state and config management.**

Kubernetes Worker Node

- **A worker node can be a virtual machine or a physical server where containers are deployed**
- **Every node in the Kubernetes cluster must run a container runtime, such as docker or rocket**
- **Two worker node components required to communicate with Kubernetes master:**
  - kubelet and
  - kubeproxy

Kubernetes Worker Node

- **Kubelet**
  - Kubelet is a primary node agent that runs on each worker node inside the cluster.
  - The primary objective of the kubelet is as follows:
    - it looks at the pods spec that was submitted to the API Server on the Kubernetes master and ensures that containers described in pods' spec are running and healthy
    - In case, kubelet notices any issues with the pods running on the worker node, then it tries to restart the pod on the same node
  - In case if the issue is with the worker node itself, then Kubernetes master detects node failure and it tries to decide to recreate the pod on another healthy node

Kubernetes Worker Node

- **Kube-Proxy**
  - Kube-Proxy essentially maintains the distributed network across all nodes, across all pods, and across all containers.
  - It also exposes services outside world

Kubernetes Worker Node

- **Pods**
  - Pod is essentially a scheduling unit in Kubernetes,
    - just like a VM in the virtualization world
  - Each pod contains one or more containers –
    - in most cases there will be only one container per pod
  - There may be scenarios where you run two or more containers inside a pod –
    - in that case one container may be helping another container
  - The primary advantage of the pod is that we can deploy multiple dependent containers together
  - It adds wrapper around these containers
  - **We interact and manage these containers primarily through pods**

Kubernetes Worker Node

- **Containers**
  - Containers provide runtime environment for applications
    - You can run containerized application processes inside the containers
  - These containers reside inside the pods on a Kubernetes cluster
  - Containers are designed to run microservices
    - Containers are not ideal for running monolithic applications!
  - Kubernetes supports docker-based containers as well as rocket-based containers

Deployment Yaml

- **Kubernetes uses YAML as a deployment file**
  - You are allowed to use JSON or YAML.
  - YAML is like JSON, but instead of curly braces you use indentation.
- **It contains declared desired state**
  - Within YAML file you can declare essentially any type of resource to be created, such as deployment, pod, service, or replication control
  - Kubernetes creates and maintains them for you

```
apiVersion: apps/v1beta1 # for versions before 1.6.0 use extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

- **This is an example of Deployment File for nginx**
  - It says that you want 3 copies of nginx-1.7.9 running on port 80 within your cluster

Deployment Yaml

- **When deploying on Kubernetes, you pair a deployment object with a service object.**

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: survey-frontend-deployment
  labels:
    app: survey-frontend
spec:
  replicas: 1
  selector:
    matchLabels:
      app: survey-frontend
  template:
    metadata:
      labels:
        app: survey-frontend
    spec:
      containers:
      - name: survey-frontend-image
        image: rdlgmu/survey-frontend:$BUILD_NUMBER
        ports:
        - containerPort: 80

---
```

```yaml
apiVersion: v1
kind: Service
metadata:
  name: survey-frontend-service
spec:
  type: LoadBalancer
  selector:
    app: survey-frontend
  ports:
  - name: http
    protocol: TCP
    port: 80
    targetPort: 8080
```

Deployment Yaml

- **Kubernetes automatically reboots pods in case of failures/crash**
  - A pathfinder resiliency feature for containers
- **Kubernetes also support rolling updates.**
  - For example, if I have version 1 deployed, and I ask it to push out version 2, it will add version 2, but it will make sure to take down version 1 only when version 2 is available,
    - so there is no downtime, even if when I am running one copy.
  - As soon as version 2 is ready, it will take down version 1 and change the service to point to version 2 on the fly.

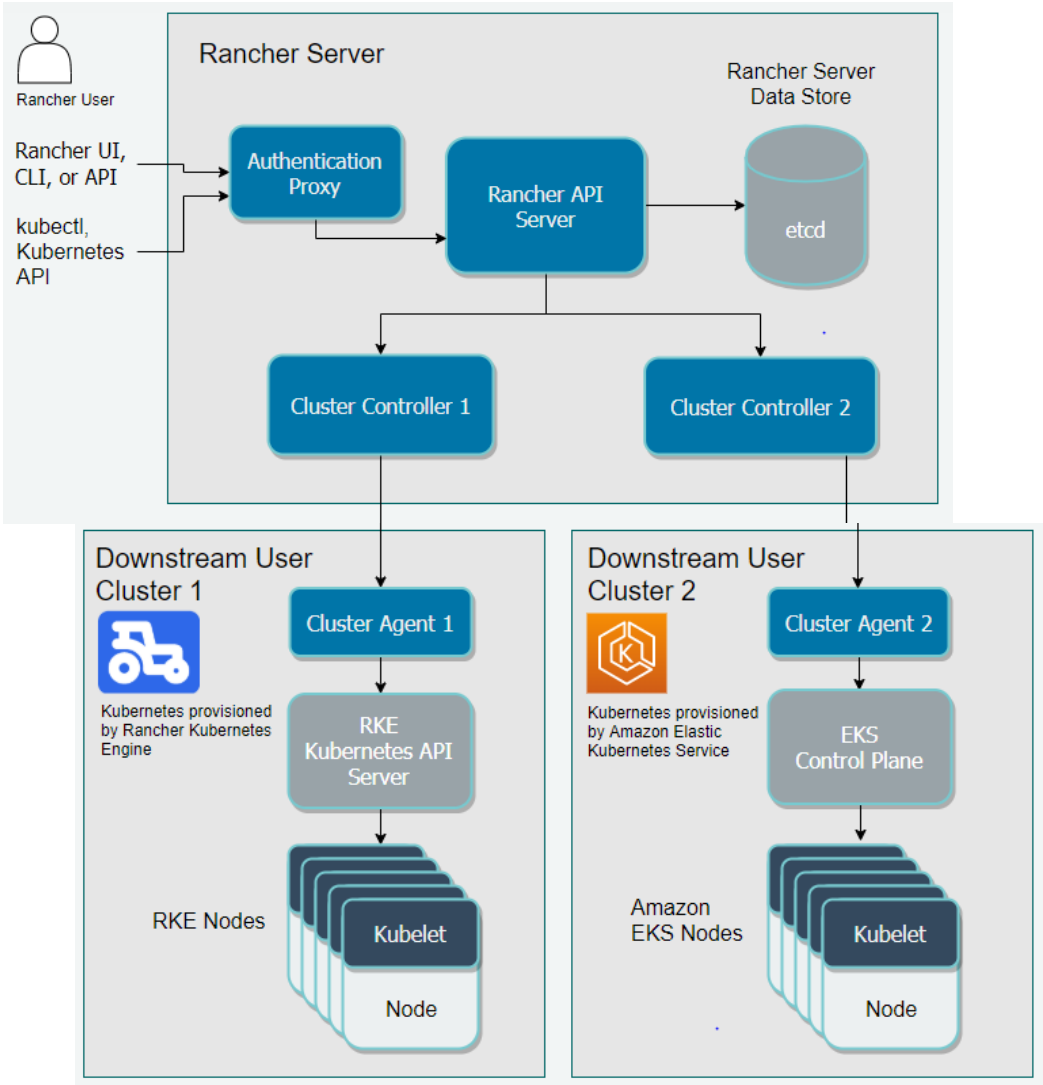
Flow to deploy on Kubernetes cluster

- **Overall Flow to deploy an application on Kubernetes cluster**
  - kubectl writes to the API Server
  - API Server validates the request and persists it to Cluster store(etcd)
  - Cluster store (etcd) notifies back the API Server
  - API Server invokes the Scheduler
  - Scheduler decides where to run the pod on and return that to the API Server
  - API Server persists it to etcd
  - etcd notifies back the API Server.
  - API Server invokes the Kubelet in the corresponding worker node
  - Kubelet talks to the Docker daemon using the API over the Docker socket to create the container
  - Kubelet updates the pod status to the API Server
  - API Server persists the new state in etcd

Rancher Kubernetes Engine (RKE)

- **Kubernetes is a rich and full featured and this makes it difficult to install and configure**
  - There are installers that simplify deploying raw K8s that provide their own API to abstract access to K8s
- **RKE, pronounced as “Rake”, is a lightweight Kubernetes installer for bare-metal and virtualized servers**
  - FOSS (free and open source) product by Rancher Labs
- **Easy configuration and startup of Kubernetes cluster**
  - Specify Node IPs and roles
  - Running cluster in minutes
- **Multiple networking options available**
  - Container Networking Interface (CNI) is an open standard for container network communication

Rancher can manage multiple Kubernetes clusters



AWS DevOps Tools: Git

- **AWS CodeCommit**
  - Amazon's source code control service that hosts private Git repositories
  - Managed, highly scalable, and secure
  - Eliminates the need to operate your own source control system
    - or worry about scaling its infrastructure

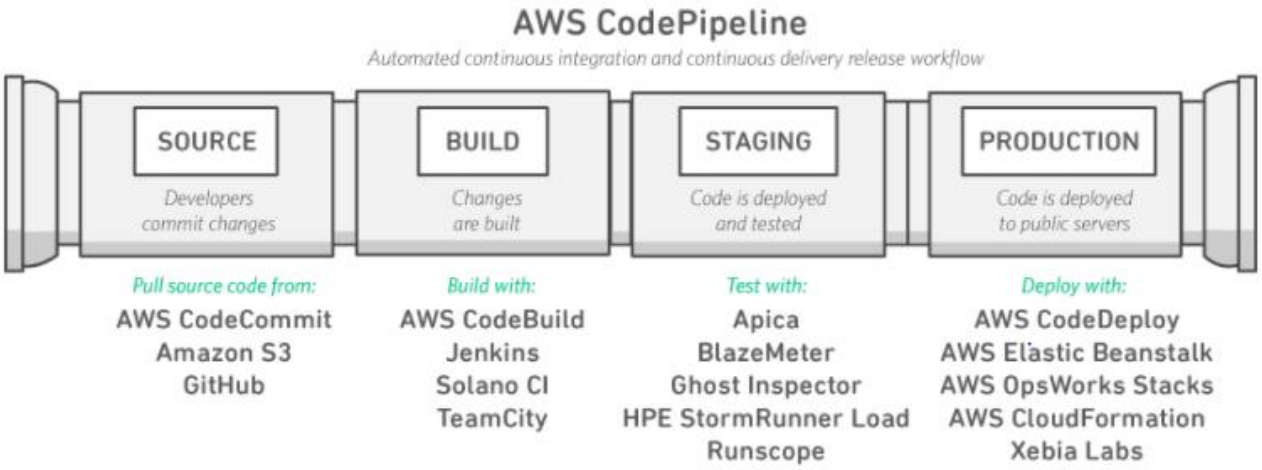
A command-line session showing repository creation, addition of a file, and remote synchronization

AWS DevOps Tools

- **AWS CodeBuild**
  - Amazon's fully managed build service
    - that compiles source code, runs tests, and produces software packages
  - Scales continuously
    - and processes multiple builds concurrently
  - You can provide custom build environments

AWS DevOps Tools

- **AWS CodePipeline**
  - Continuous delivery service for fast and reliable application updates.
  - Model and visualize your software release process.
  - Builds, tests, and deploys your code every time there is a code change.

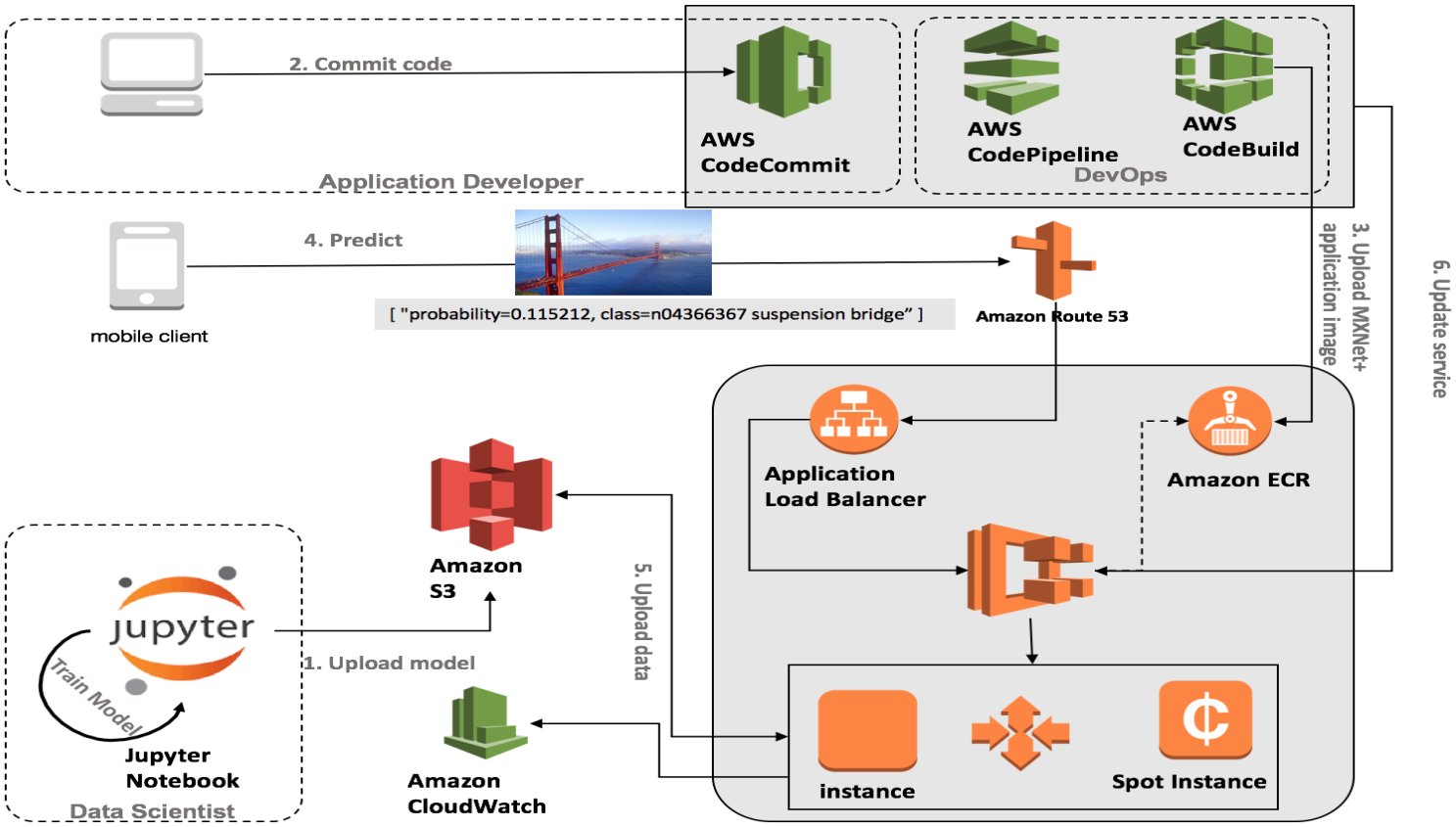


Amazon EKS (Elastic Kubernetes Service)

- **Amazon EKS is a managed Kubernetes service that makes it easy to deploy, manage, and scale containerized applications**
  - using Kubernetes on AWS
- **It runs the Kubernetes management infrastructure across multiple AZs**
  - to eliminate a single point of failure.
- **Amazon EKS is certified Kubernetes conformant**
  - so you can use existing tooling and plugins from partners and the Kubernetes community.
- **Applications running on any standard Kubernetes environment are fully compatible**
  - and can be easily migrated to Amazon EKS.
- **Amazon EKS is not free!**

CI/CD Architecture – putting it all together

- **Using AWS DevOps tools**



Backup

DevOps Tools: Ant, Maven, Gradle

- **Java build automation tools that work with Jenkins: Ant, Maven, Gradle**
- **Apache Ant**
  - Make –the first build automation tool since 1976;
    - Used for building Java applications in the early Java years.
    - Later, Ant was released as a better alternative.
  - Ant ("Another Neat Tool") is a Java library used for automating build processes for Java applications.
    - In many aspects, Ant is very similar to Make.
  - Ant build files are written in XML, and by convention, they're called build.xml.
  - Different phases of a build process are called "targets".

DevOps Tools: Ant

- **An example build.xml file for a java project with HelloWorld main class**
- This build file defines four targets:
  - *clean*, *compile*, *jar* and *run*.
  - For example, we can compile the code by running:

```
%ant compile
```

```
<project>
    <target name="clean">
        <delete dir="classes" />
    </target>

    <target name="compile" depends="clean">
        <mkdir dir="classes" />
        <javac srcdir="src" destdir="classes" />
    </target>

    <target name="jar" depends="compile">
        <mkdir dir="jar" />
        <jar destfile="jar/HelloWorld.jar" basedir="classes">
            <manifest>
                <attribute name="Main-Class"
                  value="antExample.HelloWorld" />
            </manifest>
        </jar>
    </target>

    <target name="run" depends="jar">
        <java jar="jar/HelloWorld.jar" fork="true" />
    </target>
</project>
```

Source: Google

DevOps Tools: Maven

- **Apache Maven**
  - Apache Maven is a dependency management and a build automation tool, primarily used for Java applications
  - Maven provides built-in support for dependency management
  - Maven's configuration file, containing build and dependency management instructions, is by convention called pom.xml
  - Maven also prescribes project structure

DevOps Tools: Gradle

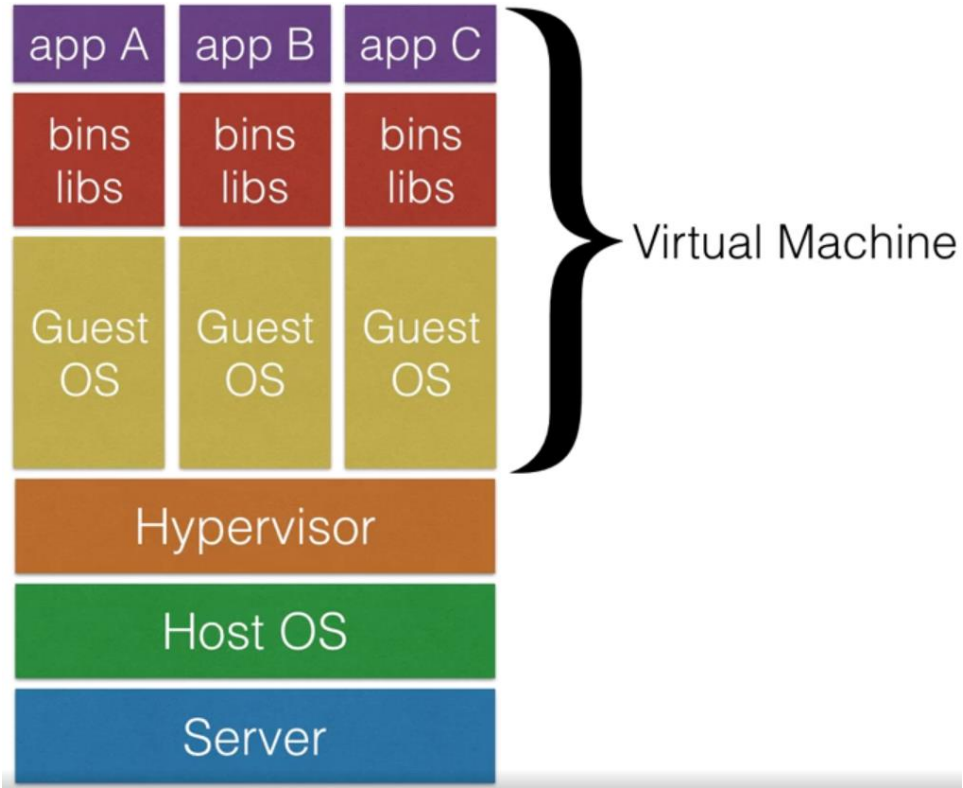
- **Apache Gradle**
  - Gradle is a dependency management and a build automation tool
    - Built upon the concepts of Ant and Maven.
  - Gradle does not use XML files, unlike Ant or Maven.
  - Gradle uses Gradle Build Language or DSL based on Groovy.
    - This led to smaller configuration files with less clutter
  - Gradle's configuration file is by convention called build.gradle.

Useful Links from youtube that you might find helpful for your Homework 2

- **Github:**
  - https://www.youtube.com/watch?v=_qcKe6nBfNE
- **Ranchers Cluster buildup:**
  - https://www.youtube.com/watch?v=jF8jCg1WPwo
- **Jenkins Installation:**
  - https://www.youtube.com/watch?v=twkRYe6m8DQ
- **Build Jenkins:**
  - https://www.youtube.com/watch?v=_MIssWPguZ0
- **Github webhook trigger:**
  - https://www.youtube.com/watch?v=xlvSjDHvUwU

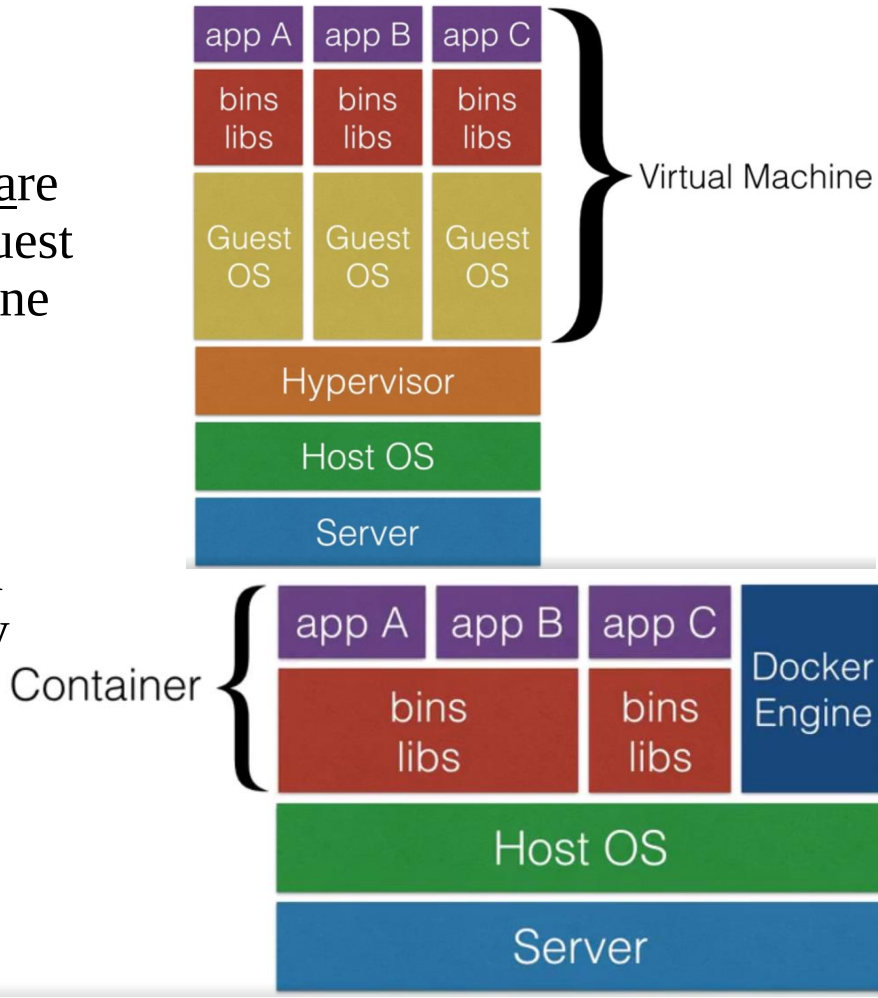
Virtual Machines

- **A virtual machine runs its own guest operating system with the allocated resources (CPU, Memory) and environment provided by the program Hypervisor**
  - **Pros**:
    - Provides a complete separation from the Host OS
  - **Cons**:
    - Need to allocate some amount of resources (CPU, RAM)
    - Hypervisor consumes some resources allocated for VMs



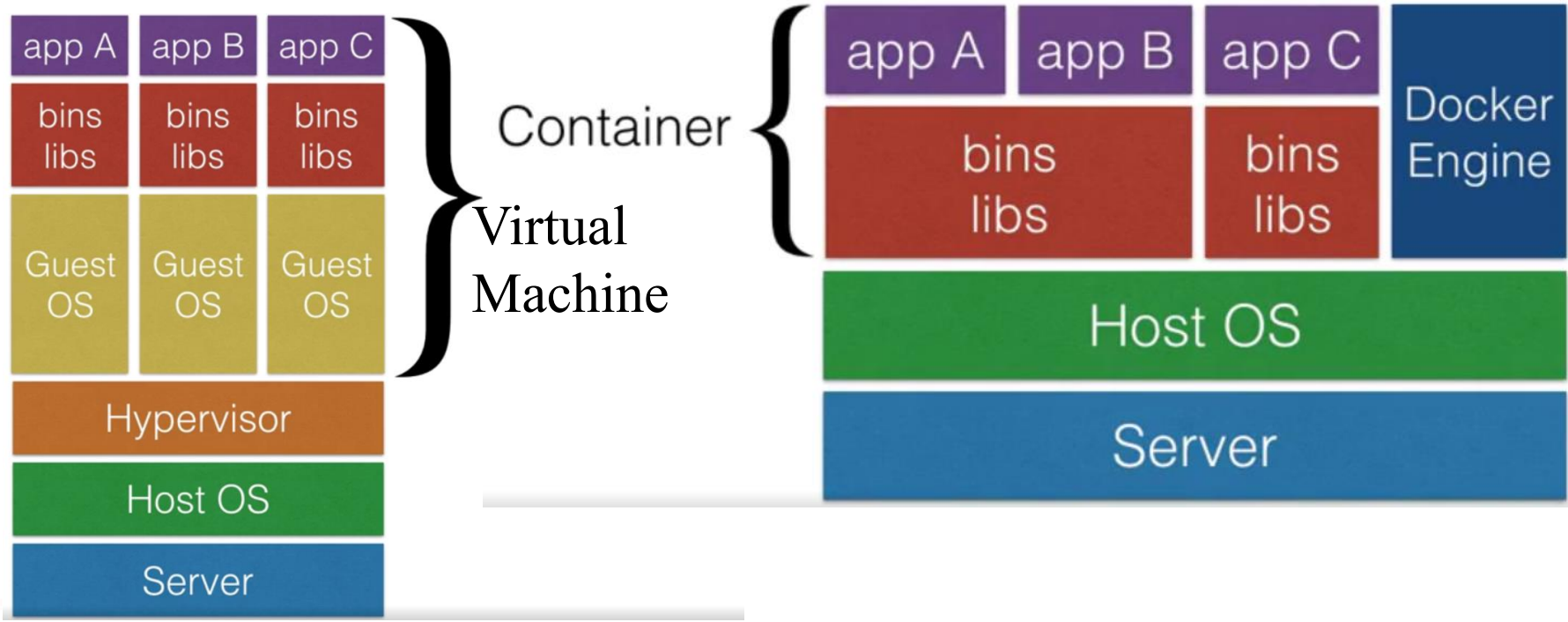
Virtual Machines vs Containers

- **Virtual machines virtualize underlying hardware**
  - The sharing and managing of hardware allows for multiple environments (guest OS) to exist on same physical machine
- **Containers allow OS level virtualization**
  - A logical packaging mechanism in which applications can be abstracted from the environments in which they run



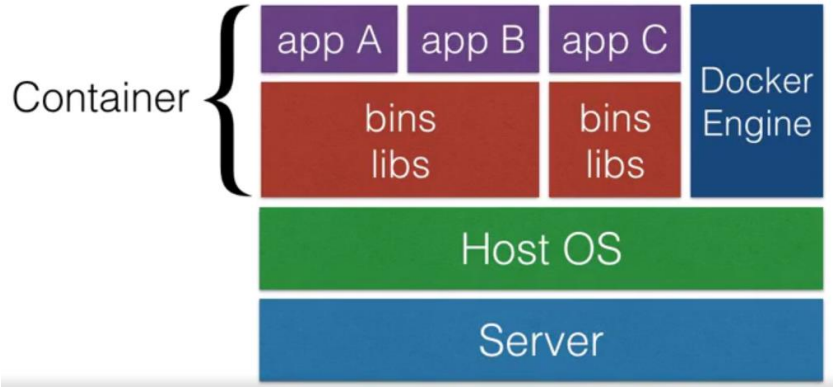
Virtual Machines vs Containers

- **Unlike Virtual Machines (VMs), containers do not bundle a full operating system**
  - Only binaries/libraries and settings required to make the software work are needed.



Benefits of Containers

- **Containers abstract the infrastructure and OS**
  - Allowing you to focus on developing business capabilities
- **An efficient, lightweight, self-contained systems**
  - Guaranteed to run the same, regardless of where it's deployed
- **Scalability, resiliency, & higher performance**
  - Without the overhead of Hypervisor



What is a Container?

- **From the direct source:**

Docker

"Containers are a way to package software in a format that can run isolated on a shared operating system. Unlike VMs, containers do not bundle a full operating system - only libraries and settings required to make the software work are needed. This makes for efficient, lightweight, self-contained systems and guarantees that software will always run the same, regardless of where it's deployed."

Source: https://www.docker.com/what-docker

Useful Links from youtube that you might find helpful for your Homework 2

- **Dockerfile**
  - A Dockerfile is a script to build images (that are used to created containers), step-by-step, layer-by-layer, automatically from a source (base) image.
  - See the following links
  - https://www.digitalocean.com/community/tutorials/docker-explained-using-dockerfiles-to-automate-building-of-images
  - https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-getting-started
  - https://rominirani.com/docker-tutorial-series-writing-a-dockerfile-ce5746617cd
  - https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices/

DevOps

- **Describes a culture and process that bring development and operations teams together**
  - to complete software development and deployment.
- **Allows organizations to create and improve products at a faster pace**
  - than they can with traditional software development approaches.