Prerequisite: Install Docker Desktop
docker.com/download

Zero to Docker

Build Night #1

Blockchain@USC Engineering Spring 2025

Blockchain@USC

Agenda

Intro to Docker

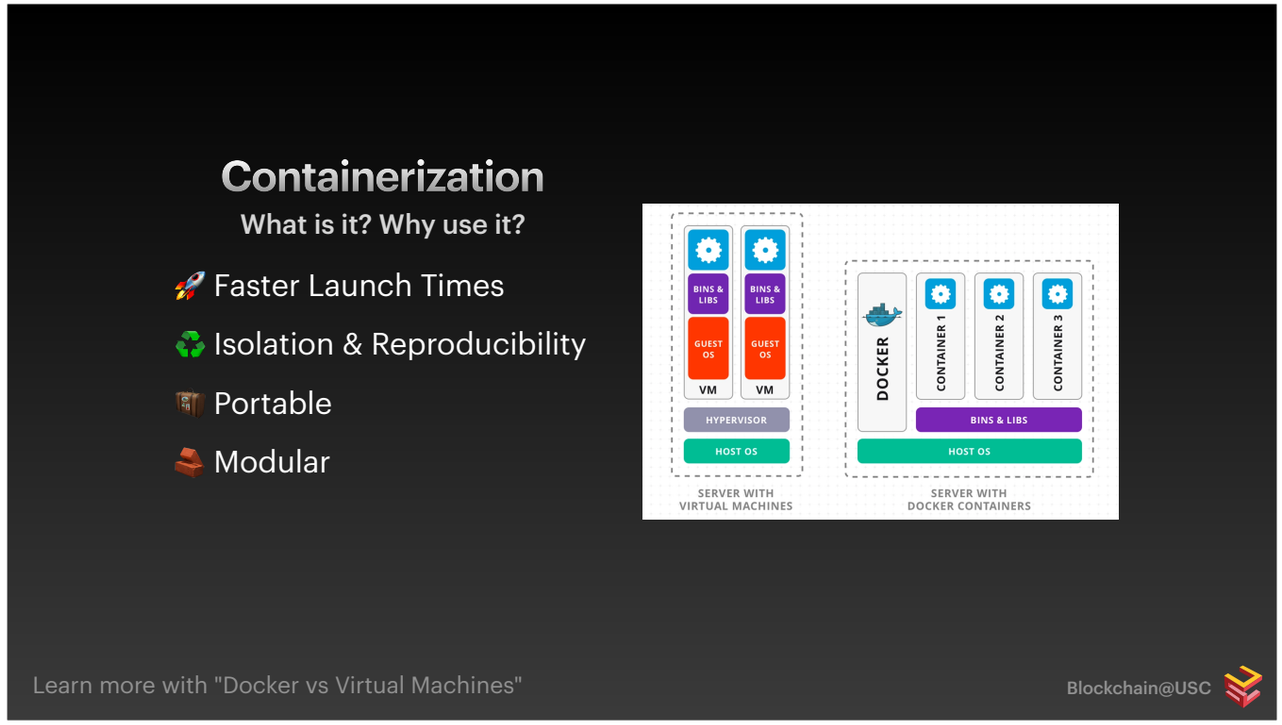
1. Overview
2. Docker Compose (Orchestrator)
3. Docker (Image, Container Spec)
4. Docker CLI (DevEx)
5. Code Demo

Placeholder: Learn More with [Websearch Query]

Blockchain@USC

Point out guides in bottom right of slide

Overview



Containerization is a lightweight way to package, ship, and run applications by encapsulating your code and dependencies into a single, isolated environment called a container. It offers several key advantages:

Containers start up much quicker than virtual machines because they share the host system's kernel instead of running their own full OS.
Each container is isolated from others on the same system, ensuring consistent environments for your applications and services.
Containers can run anywhere—on any machine or cloud platform—as long as there’s a compatible container runtime (eg. Docker, K8).
You can easily add or remove components like databases, APIs, or other services to build flexible and scalable systems.

With Docker, containerization becomes accessible and efficient, empowering you to develop, deploy, and scale applications faster than ever before.

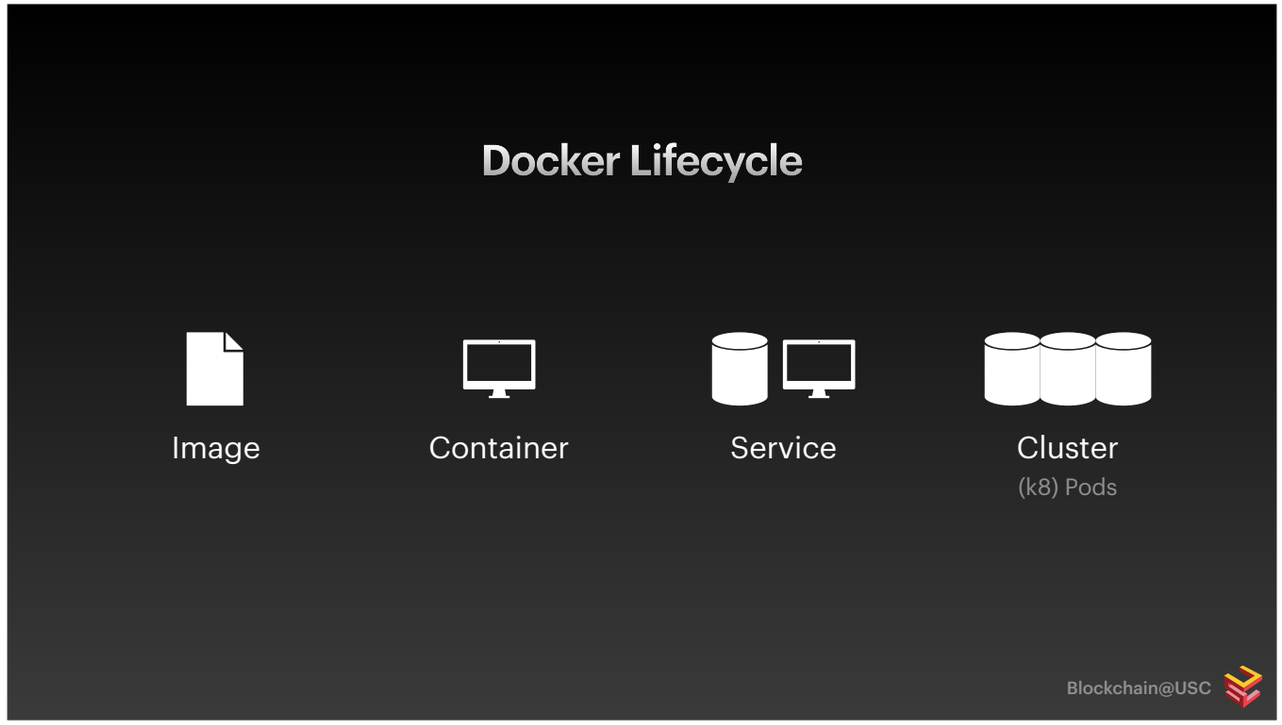


Image - Readonly template containing instructions to create a container
Container - Runtime *instance* of a Docker image – isolated process that includes the application and its dependencies
Service - Service is a higher-level configuration that defines how one or more containers should run (image, ports, volumes, environment variables, etc.).
Cluster - Collection of machines (nodes) that collectively host and manage containers/services, often coordinated by an orchestration tool (e.g., Docker Swarm or Kubernetes).

Docker Compose

Container Orchestrator

Docker Compose

YAML Syntax

Infrastructure as Code
YAML-based definition for multi-container Docker apps

Structure:

- `services`
- `networks`
- `volumes`

```yaml
version: "3.8"

services:
  # 1. Example service that extends a service defined in another docker-compose file
  extended-service:
    extends:
      service: base-service
      file: ./docker-compose.base.yml
    environment:
      - EXTENDED_ENV=overridden_value

  # 2. Service built from a public image
  public-image-service:
    image: nginx:alpine
    ports:
      - "80:80"
    networks:
      - frontend

  # 3. Service built from a local Dockerfile
  local-build-service:
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - "8080:8080"
    networks:
      - backend
    volumes:
      - mydata:/usr/src/app/data

networks:
  frontend:
    driver: bridge
  backend:
    driver: bridge

volumes:
  mydata:
```

Learn more with "Docker Compose syntax"

Blockchain@USC

Infrastructure as Code: Capture application's environment in a declarative manner, eliminating the need to manage each container separately.
Portability of application infrastructure to migrate to cloud providers
`.env` is default file, but you can pass unique env variables to each service
useful for staging environments `.env.dev` vs `.env`

```yaml
services:
  # 1. Example service that extends a service defined in another docker-compose file
  extended-service:
    extends:
      service: base-service
      file: ./docker-compose.base.yml
    environment:
      - EXTENDED_ENV=overridden_value

  # 2. Service built from a public image
  public-image-service:
    image: nginx:alpine
    ports:
      - "80:80"
    networks:
      - frontend

  # 3. Service built from a local Dockerfile
  local-build-service:
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - "8080:8080"
    networks:
      - backend
    volumes:
      - mydata:/usr/src/app/data
```

Learn more

chain@USC

Deeper look at the YAML syntax (3 fundamental approaches):

1. Chain multiple docker-compose files (if we want to create abstract wrappers around microservices)
2. Build from public images found on the Docker Hub marketplace
3. Define custom Dockerfile logic and specify container specs (Dockerfiles discussed later)

Docker Compose

Command Line Interface (CLI)

- **Build & Run**
  `build, up, start`
- **Monitor**
  `logs, ps, exec`
- **Stop & Remove**
  `stop, rm, down`

```
Commands:
  attach      Attach local standard input, output, and error streams to a servi
  build       Build or rebuild services
  commit      Create a new image from a service container's changes
  config      Parse, resolve and render compose file in canonical format
  cp          Copy files/folders between a service container and the local file
  create      Creates containers for a service
  down        Stop and remove containers, networks
  events      Receive real time events from containers
  exec        Execute a command in a running container
  export      Export a service container's filesystem as a tar archive
  images      List images used by the created containers
  kill        Force stop service containers
  logs        View output from containers
  ls          List running compose projects
  pause       Pause services
  port        Print the public port for a port binding
  ps          List containers
  pull        Pull service images
  push        Push service images
  restart     Restart service containers
  rm          Removes stopped service containers
  run         Run a one-off command on a service
  scale       Scale services
  start       Start services
  stats       Display a live stream of container(s) resource usage statistics
  stop        Stop services
  top         Display the running processes
  unpause     Unpause services
  up          Create and start containers
  version     Show the Docker Compose version information
  wait        Block until containers of all (or specified) services stop.
  watch       Watch build context for service and rebuild/refresh containers wh

Run 'docker compose COMMAND --help' for more information on a command.
```

Learn more with "`docker compose [cmd] --help`"

Blockchain@USC

up vs **start**: docker compose up will create the service, start requires that the service is already created

Docker Compose

Common Commands

`docker compose -f <path> [cmd]`
Specify file path for docker compose commands

`docker compose up -d [service...]`
Create and start containers in detached mode; omitting services will start all defined services

`docker compose exec -it <service> /bin/bash`
Open an interactive shell to troubleshoot in a service's container

`docker compose logs -f [service...]`
Display logs from a service (and follow stdout)

```
Commands:
  attach      Attach local standard input, output, and error streams to a servi
  build       Build or rebuild services
  commit      Create a new image from a service container's changes
  config      Parse, resolve and render compose file in canonical format
  cp          Copy files/folders between a service container and the local file
  create      Creates containers for a service
  down        Stop and remove containers, networks
  events      Receive real time events from containers
  exec        Execute a command in a running container
  export      Export a service container's filesystem as a tar archive
  images      List images used by the created containers
  kill        Force stop service containers
  logs        View output from containers
  ls          List running compose projects
  pause       Pause services
  port        Print the public port for a port binding
  ps          List containers
  pull        Pull service images
  push        Push service images
  restart     Restart service containers
  rm          Removes stopped service containers
  run         Run a one-off command on a service
  scale       Scale services
  start       Start services
  stats       Display a live stream of container(s) resource usage statistics
  stop        Stop services
  top         Display the running processes
  unpause     Unpause services
  up          Create and start containers
  version     Show the Docker Compose version information
  wait        Block until containers of all (or specified) services stop.
  watch       Watch build context for service and rebuild/refresh containers wh

Run 'docker compose COMMAND --help' for more information on a command.
```

Learn more with "`docker compose [cmd] --help`"

Blockchain@USC

good practice to organize microservices orchestrations in separate docker compose files

also check out **profiles** in docker-compose syntax to categorize services

detached mode means that the a new process is created on the host OS (ie. runs in the bkgd)

Dockerfile

Container Configuration

Blockchain@USC

Quick checkin with audience

Dockerfile

Layer-by-Layer Construction

Instruction-by-Instruction Build
Each Dockerfile instruction creates a new layer in the final image

Immutable Layers
Layers are cached if no change is detected in that instruction or prior layers

```
# ----------------------------------------
# Stage 1: Build stage
# ----------------------------------------
# Use an official Node.js image for building
FROM node:16 AS build

# Example build argument with default value
ARG BUILD_ENV=development

# Set environment variable for the build process
ENV NODE_ENV=$BUILD_ENV

# Set the working directory
WORKDIR /app

# Copy package.json and package-lock.json
COPY package*.json ./

# Install dependencies
RUN npm install

# Copy remaining project files to the container
COPY . .

# Run a build script (for example, building a React or Vue app)
RUN npm run build

```

Learn more with "Dockerfile syntax"

Blockchain@USC

Assembly-Like: Each line is a directive that sequentially builds the final image.

Declarative and Reproducible: Ensure the same build output regardless of environment or host.

Dockerfile

Instruction Set

`FROM <img>:<version>`
Base image declaration

`RUN <cmd> <arg...>`
Execute commands (like installing packages)

`COPY / ADD <src> <dst>`
Copy files from host to container's filesystem

`ENV <key> <value>`
Set environment variables within image

`WORKDIR <dir>`
Set the working directory for subseq. actions

`EXPOSE <port>`
Documents container ports

`CMD / ENTRYPOINT <cmd> <arg...>`
Define default command / process

```
# ----------------------------------------
# Stage 1: Build stage
# ----------------------------------------
# Use an official Node.js image for building
FROM node:16 AS build

# Example build argument with default value
ARG BUILD_ENV=development

# Set environment variable for the build process
ENV NODE_ENV=$BUILD_ENV

# Set the working directory
WORKDIR /app

# Copy package.json and package-lock.json
COPY package*.json ./

# Install dependencies
RUN npm install

# Copy remaining project files to the container
COPY . .

# Run a build script (for example, building a React or Vue app)
RUN npm run build

```

Learn more with "Dockerfile syntax"

Dockerfile

Best Practices

Minimize Layers
Combine related RUN steps to optimize image size

Leverage Caching
Keep frequently changed instructions toward the end to avoid invalidating earlier layers

Use Official Base Images
Get security updates and runtime stability from trusted sources

```
# ----------------------------------------
# Stage 1: Build stage
# ----------------------------------------
# Use an official Node.js image for building
FROM node:16 AS build

# Example build argument with default value
ARG BUILD_ENV=development

# Set environment variable for the build process
ENV NODE_ENV=$BUILD_ENV

# Set the working directory
WORKDIR /app

# Copy package.json and package-lock.json
COPY package*.json ./

# Install dependencies
RUN npm install

# Copy remaining project files to the container
COPY . .

# Run a build script (for example, building a React or Vue app)
RUN npm run build

```

Learn more with "Dockerfile Best Practices"

Dockerfile

Multi-Stage Builds

Build Image v Final Image
Isolate heavy build tools in the first stage, then copy only what's needed into a lean production stage

Reduced Final Size
Avoid shipping compilers or unnecessary dependencies in the final container

```
# ----------------------------------------------------
# Stage 1: Build stage
# ----------------------------------------------------
# Use an official Node.js image for building
FROM node:16 AS build

# Example build argument with default value
ARG BUILD_ENV=development

# Set environment variable for the build process
ENV NODE_ENV=$BUILD_ENV

# Set the working directory
WORKDIR /app

# Copy package.json and package-lock.json
COPY package*.json ./

# Install dependencies
RUN npm install

# Copy remaining project files to the container
COPY . .

# Run a build script (for example, building a React or Vue app)
RUN npm run build

# ----------------------------------------------------
# Stage 2: Production stage
# ----------------------------------------------------
# Use a lightweight Nginx image to serve the built app
FROM nginx:alpine

# Example environment variables for Nginx
ENV NGINX_HOST=0.0.0.0
ENV NGINX_PORT=80

# Copy the build artifacts from the first stage
COPY --from=build /app/build /usr/share/nginx/html

# Expose a port (primarily for documentation - actual exposure depends on docker run/compose)
EXPOSE 80

# Define a volume for Nginx cache (or logs)
VOLUME ["/var/cache/nginx"]

# ENTRYPOINT sets the container's main process
ENTRYPOINT ["nginx"]

# CMD provides default arguments to the ENTRYPOINT
CMD ["-g", "daemon off;"]
```

Learn more with "Dockerfile Multistage Builds"

Blockchain@USC

Point out how stages are named with FROM instruction

```
24
25  # Run a build script (for example, building a React or Vue app)
26  RUN npm run build
27
28  # ----------------------------------------
29  # Stage 2: Production stage
30  # ----------------------------------------
31  # Use a lightweight Nginx image to serve the built app
32  FROM nginx:alpine
33
34  # Example environment variables for Nginx
35  ENV NGINX_HOST=0.0.0.0
36  ENV NGINX_PORT=80
37
38  # Copy the build artifacts from the first stage
39  COPY --from=build /app/build /usr/share/nginx/html
40
41  # Expose a port (primarily for documentation - actual exposure depends on docker run/compose)
42  EXPOSE 80
43
44  # Define a volume for Nginx cache (or logs)
45  VOLUME ["/var/cache/nginx"]
46
47  # ENTRYPOINT sets the container's main process
48  ENTRYPOINT ["nginx"]
49
50  # CMD provides default arguments to the ENTRYPOINT
51  CMD ["-g", "daemon off;"]
52
```


How to pass filesystem data between builds

Dockerfile

COPY / ADD Syntax

COPY / ADD <src> <dst>
Copy files from host to container's filesystem

- **COPY dir/ /app/**
  Copies everything in dir/ into /app/
  - /app/item1
    /app/item2
- **COPY dir /app/**
  Copies directory dir into /app/
- **COPY** can copy from staged builds
  ADD allow <src> to be URL

```
# ----------------------------------------
# Stage 1: Build stage
# ----------------------------------------
# Use an official Node.js image for building
FROM node:16 AS build

# Example build argument with default value
ARG BUILD_ENV=development

# Set environment variable for the build process
ENV NODE_ENV=$BUILD_ENV

# Set the working directory
WORKDIR /app

# Copy package.json and package-lock.json
COPY package*.json ./

# Install dependencies
RUN npm install

# Copy remaining project files to the container
COPY . .

# Run a build script (for example, building a React or Vue app)
RUN npm run build

```

Learn more with "Dockerfile COPY vs ADD"

Blockchain@USC

Highlight semantics for COPY / ADD instructions

Docker CLI

Command Line Interface

Blockchain@USC

Audience Check-In
Previous slides were about defining infrastructure

Next will be about managing runtime environment using the CLI (DevEx)

Docker CLI

Core Tools

- **Build & Run**
  `build, run, push, pull`
- **Monitor**
  `logs, ps, exec, inspect`
- **Stop & Remove**
  `stop, rm` (containers),
  `rmi` (images)

```
Usage:  docker [OPTIONS] COMMAND

A self-sufficient runtime for containers

Common Commands:
  run         Create and run a new container from an image
  exec        Execute a command in a running container
  ps          List containers
  build       Build an image from a Dockerfile
  pull        Download an image from a registry
  push        Upload an image to a registry
  images      List images
  login       Authenticate to a registry
  logout      Log out from a registry
  search      Search Docker Hub for images
  version     Show the Docker version information
  info        Display system-wide information

Management Commands:
  ai*         Ask Gordon - Docker Agent
  builder     Manage builds
  buildx*     Docker Buildx
  checkpoint  Manage checkpoints
  compose*    Docker Compose
  container   Manage containers
  context     Manage contexts
  debug*      Get a shell into any image or container
  desktop*    Docker Desktop commands (Beta)
  dev*        Docker Dev Environments
  extension*  Manages Docker extensions
  feedback*   Provide feedback, right in your terminal!
  image       Manage images
  init*       Creates Docker-related starter files for your project
  manifest    Manage Docker image manifests and manifest lists
  network     Manage networks
  plugin      Manage plugins
  sbom*       View the packaged-based Software Bill Of Materials (SBOM) for an image
  scout*      Docker Scout
  system      Manage Docker
  trust       Manage trust on Docker images
  volume      Manage volumes
```

Learn more with "`docker --help`"

Docker CLI

Common Commands

`docker build -t <tag> .`
Builds an image from a Dockerfile from the current directory

`docker run -d -p <host>:<container> <image>`
Runs a container from an image, mapping ports, in detached mode

`docker exec -it <container> /bin/bash`
Open an interactive shell to troubleshoot in a running container

`docker logs -f <container>`
Display logs from a container (and follow stdout)

```
Usage:  docker [OPTIONS] COMMAND

A self-sufficient runtime for containers

Common Commands:
  run         Create and run a new container from an image
  exec        Execute a command in a running container
  ps          List containers
  build       Build an image from a Dockerfile
  pull        Download an image from a registry
  push        Upload an image to a registry
  images      List images
  login       Authenticate to a registry
  logout      Log out from a registry
  search      Search Docker Hub for images
  version     Show the Docker version information
  info        Display system-wide information

Management Commands:
  ai*         Ask Gordon - Docker Agent
  builder     Manage builds
  buildx*     Docker Buildx
  checkpoint  Manage checkpoints
  compose*    Docker Compose
  container   Manage containers
  context     Manage contexts
  debug*      Get a shell into any image or container
  desktop*    Docker Desktop commands (Beta)
  dev*        Docker Dev Environments
  extension*  Manages Docker extensions
  feedback*   Provide feedback, right in your terminal!
  image       Manage images
  init*       Creates Docker-related starter files for your project
  manifest    Manage Docker image manifests and manifest lists
  network     Manage networks
  plugin      Manage plugins
  sbom*       View the packaged-based Software Bill Of Materials (SBOM) for an image
  scout*      Docker Scout
  system      Manage Docker
  trust       Manage trust on Docker images
  volume      Manage volumes
```

Learn more with "`docker <cmd> --help`"

Code Demo

github.com/BlockchainUSC/docker-workshop