

Docker, Deployments, and CI/CD

Aditya Putra

me@adityaputra.com

<https://github.com/adityaputra/praktisimengajar-cloudcomputing>

What we'll be learning

- Module 1: Introduction to Docker
 - Module 2: Docker Images and Containers
 - Module 3: Docker Networking and Volumes
 - Module 4: Docker Compose
 - Module 5: Deploying Docker Containers to AWS
 - Module 6: Alternative Deployment Methods and CI/CD with AWS CodeDeploy
-

Introduction to Docker

Aditya Putra

me@adityaputra.com

<https://github.com/adityaputra/praktisimengajar-cloudcomputing>

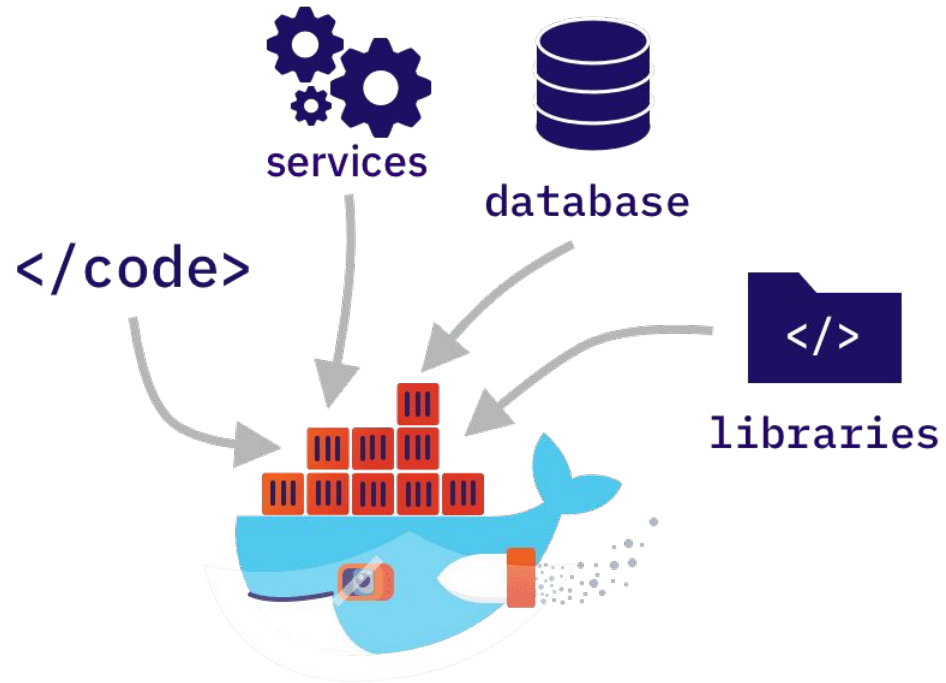
Module 1

Introduction to Docker

- What is Docker and why is it important?
- Key concepts: images, containers, Dockerfile, Docker Hub
- Installing Docker and running your first container

What is Docker?

Docker is an open-source platform that enables you to automate the deployment, scaling, and management of applications using containerization.



Why is it important?

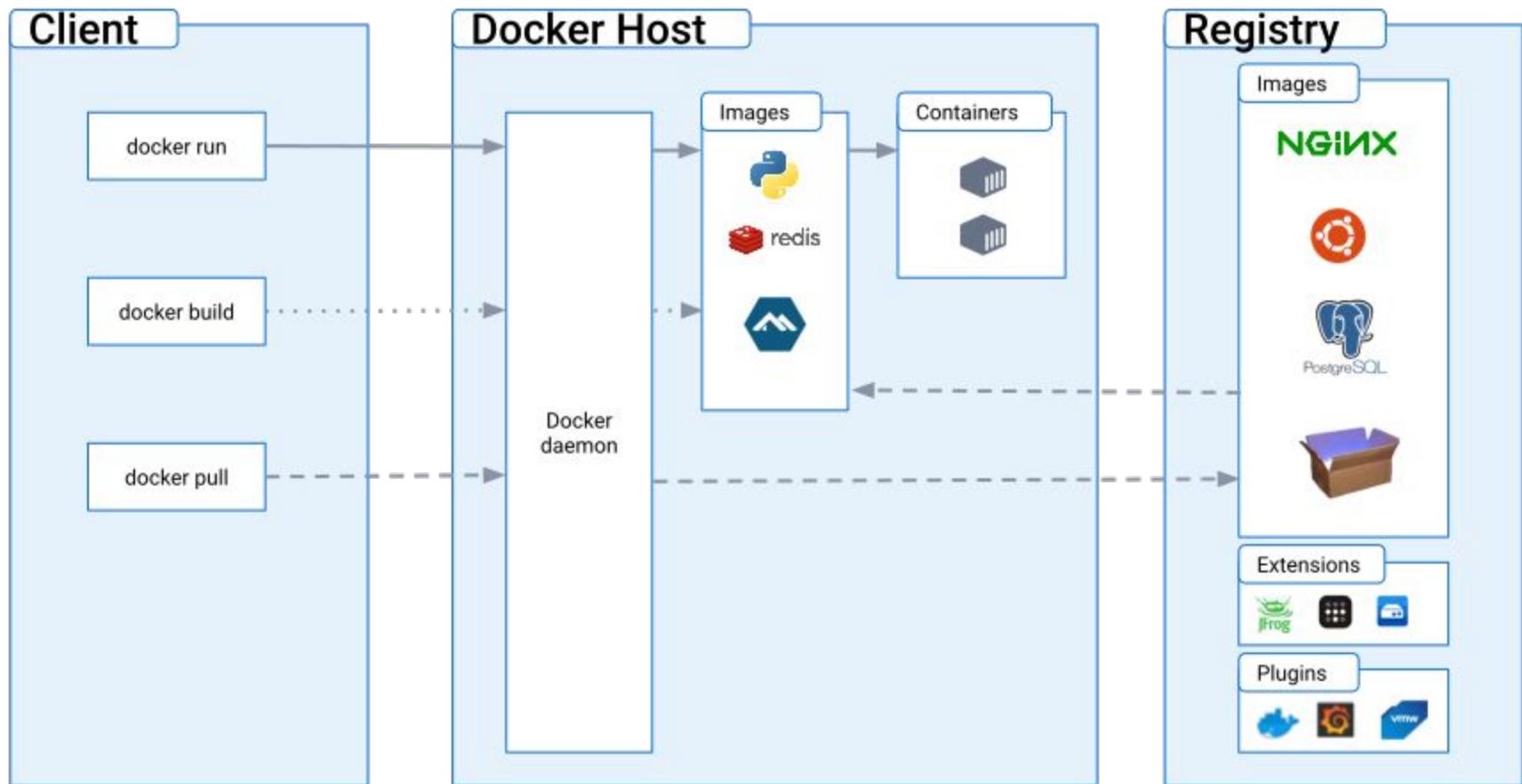
- **Consistent** environments: Containers provide a consistent and reproducible environment for applications, eliminating the "works on my machine" problem.
- Resource **efficiency**: Containers share the host system's OS kernel, resulting in lower resource usage compared to traditional virtualization.
- **Scalability**: Docker allows you to easily scale applications by running multiple containers across different hosts.
- **Isolation**: Containers provide isolation between applications, ensuring that they do not interfere with each other.

Containerization and cloud

- Flexibility and portability
- Rapid deployment
- Scalability and resource efficiency
- DevOps practices

Key concepts

- **Images:** Docker images are read-only templates that define the application's code, runtime, libraries, and dependencies. They serve as the basis for creating containers.
- **Containers:** Docker containers are instances of Docker images. They encapsulate the application and its dependencies, enabling consistent execution across different environments.
- **Dockerfile:** A Dockerfile is a text file that contains instructions for building a Docker image. It defines the base image, required packages, environment variables, and commands to run during container creation.
- **Docker Hub:** Docker Hub is a cloud-based **registry** that hosts public and private Docker images. It allows you to easily share and distribute Docker images with others.



Running your first container

- Pulling image from docker registry (default is docker hub)
 - `docker pull nginx:latest`
- Running a container (you can also run container directly and docker will automatically pull the image for you if it's not in your local)
 - `docker run -d -p 80:80 nginx`
 - `docker run hello-world`

Installing docker

- Install: <https://docs.docker.com/get-docker/>
- Awesome collection of docker-related resources:
<https://awesome-docker.netlify.app/>

Managing your containers

- `docker ps`
 - `docker ps -a`
- `docker stop`
- `docker start`
- `docker rm`

Demo

Conclusion

Docker Images and Containers

Aditya Putra

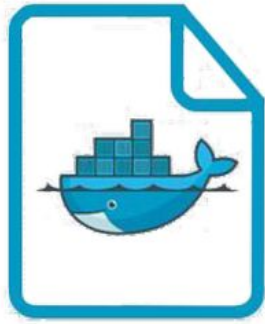
me@adityaputra.com

<https://github.com/adityaputra/praktisimengajar-cloudcomputing>

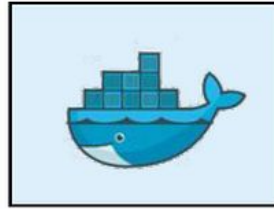
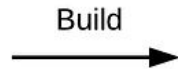
Module 2

- Creating Docker images using Dockerfiles
- Building images and tagging them
- Managing containers: starting, stopping, and removing containers
- Managing images: pulling, pushing, and deleting images from Docker Hub

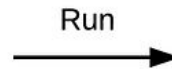
Dockerfiles



Dockerfile



Docker
Image



Docker
Container

```
#dependencies
FROM node:alpine

#working directory
WORKDIR /app

#copying dependency for our app into working directory
COPY src/package.json .

# command to run to install and configure app's dependencies
RUN npm install

# copy everything else into workdir
COPY src/. .

# exposing port for the app
EXPOSE 1000

# command to run our app
CMD ["node", "index.js"]
```



Building images and tagging them

- Building images from Dockerfiles using the **docker build** command.
- Tagging images with a specific version or label using the **-t** or **--tag** flag.
- Best practices for tagging images, such as using semantic versioning.

Demo

Managing containers

- Starting containers with the **docker run** command.
- Specifying container names, ports, and environment variables with flags.
- Stopping containers with the **docker stop** command.
- Removing stopped containers with the **docker rm** command.

Demo

Managing images

- Pulling images from Docker Hub using the **docker pull** command.
- Pushing images to Docker Hub using the **docker push** command.
- Managing images locally: listing images, inspecting image details, and deleting images.

Demo

Exploring docker hub and other registries

- <https://hub.docker.com/>
- <https://aws.amazon.com/ecr/>
- <https://ghcr.io>
- Many others

Demo

Assignment

- Create a docker image for a simple web app and publish the image on docker hub.
 - You can use any programming language of your choice
 - Be sure that people can simply run **docker pull** and **docker run** using your image name on docker hub
- Provide the docker image URL to us for submission

Docker Networking and Volumes

Aditya Putra

me@adityaputra.com

<https://github.com/adityaputra/praktisimengajar-cloudcomputing>

Module 3

Docker Networking and Volumes

- Docker networking basics
- Docker network types
- Creating and managing Docker networks
- Docker volumes basics
- Creating and managing Docker volumes
- Using volumes for data persistence

Docker networking basics

- Default network: bridge
- Container IP addresses
- Container DNS resolution

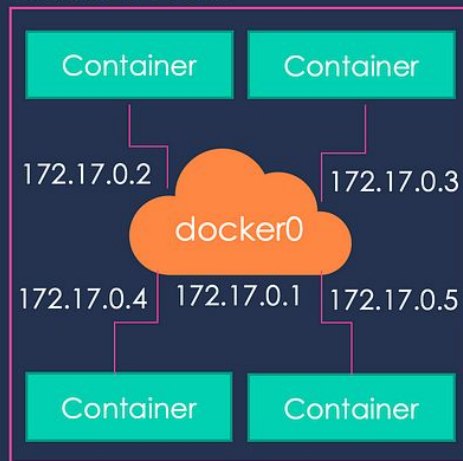
Docker network types

- Bridge network: The default network that allows containers on the same host to communicate with each other using IP addresses.
- Host network: Containers use the host's network stack, sharing the host's IP address and network interfaces.
- Overlay network: Used for multi-host communication, allowing containers on different hosts to communicate securely.
- None: network is disabled

Bridge

`docker run ubuntu`

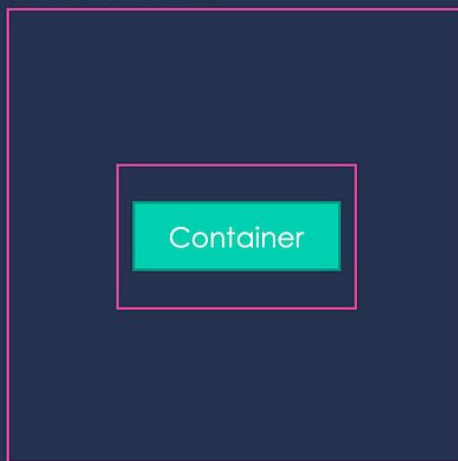
Docker Host



None

`docker run \`
`--network=none`
`ubuntu`

Docker Host



Host

`docker run \`
`--network=host`
`ubuntu`

Docker Host



Creating and managing Docker networks

- Creating user-defined networks using the **docker network create** command.
- Connecting containers to networks using the **--network** flag when running containers.
- Inspecting network details using the **docker network inspect** command.
- Removing unused networks with the **docker network prune** command.

Demo

Docker volumes

- Volumes: Persistent data storage mechanisms in Docker that can be shared between containers and survive container restarts.
- Volume types:
 - named volumes
 - bind-mounts / host-mounted volumes
 - anonymous volumes, etc

Creating and managing Docker volumes

- Creating named volumes using the **docker volume create** command.
- Mounting volumes to containers using the **--volume** or **-v** flag when running containers.
- Inspecting volume details using the **docker volume inspect** command.
- Listing and removing unused volumes with the **docker volume prune** command.

Using volumes for data persistence

- Mapping directories from the host to containers using host-mounted volumes.
- Sharing volumes between containers in the same Docker network.
- Backing up and restoring Docker volumes.

Demo

Docker Compose

Aditya Putra

me@adityaputra.com

<https://github.com/adityaputra/praktisimengajar-cloudcomputing>

Module 4

- Introduction to Docker Compose
- Writing a Docker Compose file
- Docker Compose commands
- Scaling services with Docker Compose
- Environment-specific configurations with Docker Compose
- Docker context

Introduction

- Docker Compose: A tool for defining and running multi-container Docker applications using a YAML configuration file.
- Benefits of Docker Compose: Simplifies container management, automates container setup, and allows defining complex multi-container environments.

Installing

<https://docs.docker.com/compose/install/>

Docker compose file

```
services:
  frontend:
    image: awesome/webapp
    ports:
      - "443:8043"
    networks:
      - front-tier
      - back-tier
    configs:
      - httpd-config
    secrets:
      - server-certificate

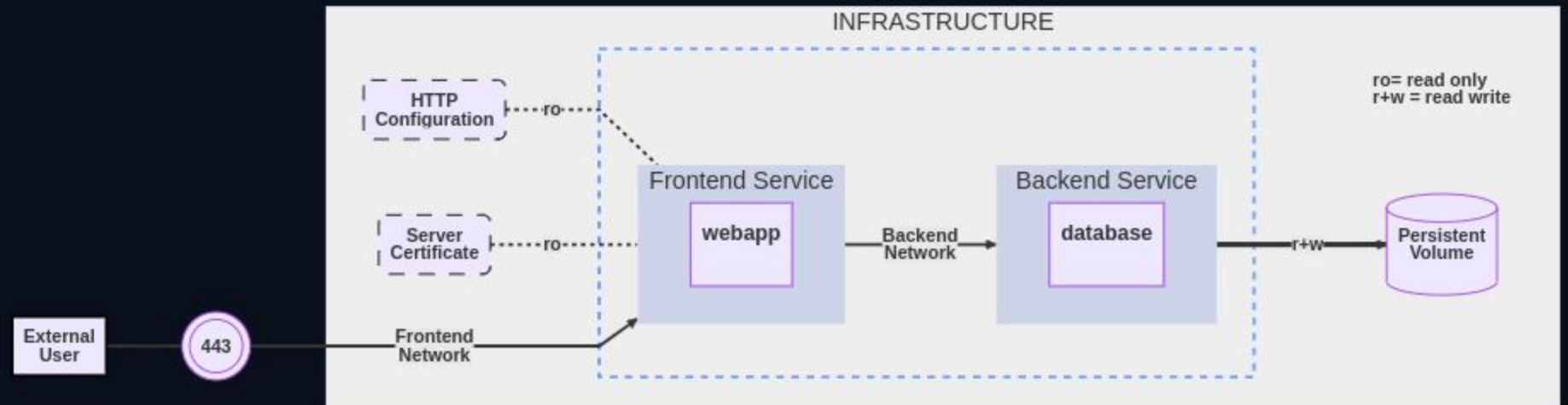
  backend:
    image: awesome/database
    volumes:
      - db-data:/etc/data
    networks:
      - back-tier

volumes:
  db-data:
    driver: flocker
    driver_opts:
      size: "10GiB"

configs:
  httpd-config:
    external: true

secrets:
  server-certificate:
    external: true

networks:
  # The presence of these objects is sufficient to define them
  front-tier: {}
  back-tier: {}
```



Docker compose commands

- **docker-compose up:** Starts containers defined in the Compose file.
- **docker-compose down:** Stops and removes containers created by the Compose file.
- **docker-compose build:** Builds or rebuilds container images defined in the Compose file.
- **docker-compose logs:** Displays logs from containers.
- **docker-compose exec:** Runs a command in a running container.

Scaling services with Docker Compose

- Scaling services horizontally using the **--scale** flag.
- Load balancing across replicated containers.

Environment-specific configs with Docker Compose

- Using environment variables in the Compose file for flexible configuration management.
- Managing different environment configurations for development, staging, and production.

Demo

Bonus

Some good self-hosted apps that you can simply docker-compose

- <https://github.com/awesome-selfhosted/awesome-selfhosted>
- <https://github.com/Haxxnet/Compose-Examples>
- <https://github.com/docker/awesome-compose>

Aditya Putra

me@adityaputra.com

<https://github.com/adityaputra/praktisimengajar-cloudcomputing>

Module 5

•

Aditya Putra

me@adityaputra.com

<https://github.com/adityaputra/praktisimengajar-cloudcomputing>

Module 6

•

The hypothesis (or prediction)

What do you think will
happen?

Research

Explain all of the research you've done about this issue/challenge.

What was the goal of your research? Be sure to explain how you found it and anyone who might have helped you!

My testing method

Each scientist uses different methods of experimentation

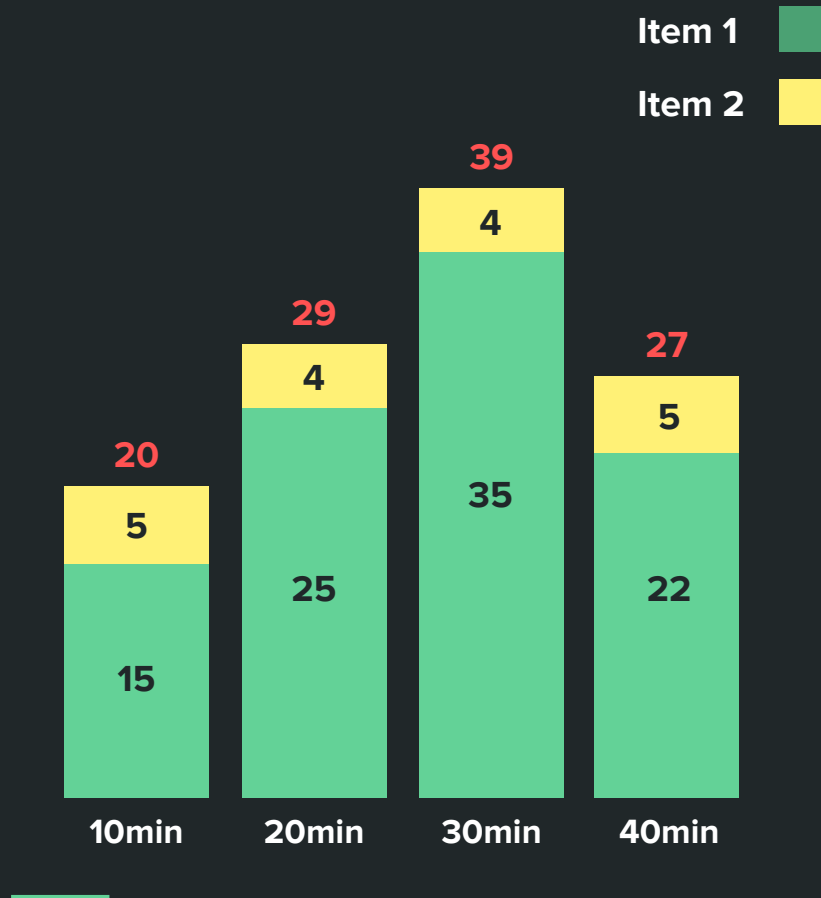
What methods did you use in your experiment?

- Lorem ipsum dolor sit amet, consectetur adipiscing elit
- Incididunt ut labore et dolore
- Consectetur adipiscing elit, sed do eiusmod tempor incididunt

Experiment data

Record the information you get from your experiment

Include a table or graph to display what you see

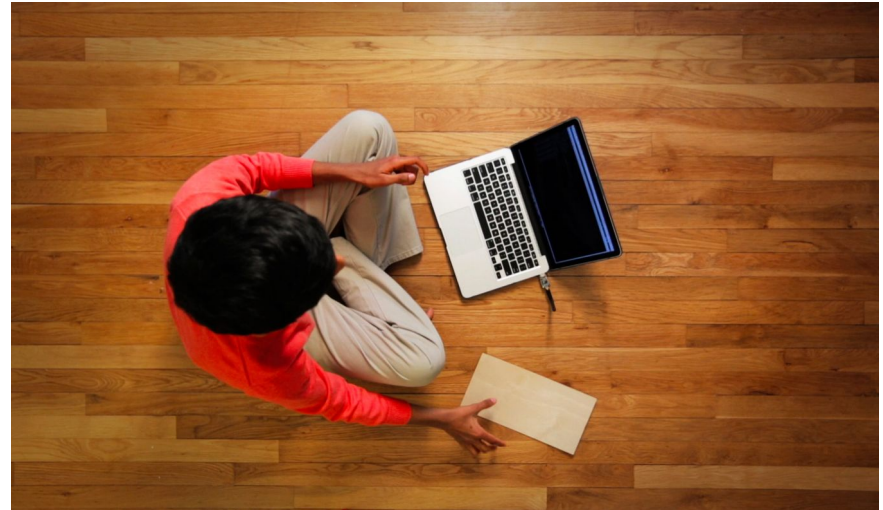


Aha!

My discoveries

What did you learn after testing?

1. Lorem ipsum dolor sit amet, consectetur adipiscing elit
2. Incidunt ut labore et dolore
3. Consectetur adipiscing elit, sed do eiusmod tempor incididunt



This is the most
important takeaway
that everyone has to
remember.

Conclusion

What is the conclusion of your experiment? Did the results support your hypothesis or predicted outcome? How will your findings help the area of science you've researched?

What will I do next?

What will you do with your findings next? How will you further your research/findings?

