Marked Assignment 2 (50%) Automation&Orchestration

Objective:

You will write python programs that allow you to automate and orchestrate microservices in the cloud. This assignment will test your ability to interact with Docker and Kubernetes, as well as your general Python scripting skills. You can use free tier on AWS in all cases to host Docker/Kubernetes servers, it should not be required to use any paid services. Your solution should be able to run on AWS EC2 instances configured as shown in the lectures.

Docker (50%):

Requirements:

1. You will write a python program for interacting with Docker, using the **Docker Python SDK** wherever possible. You may choose how the user interacts with the program, but it must provide the following basic functionality:

    - List all containers, including stopped/exited containers.
    - Run a container, with user specifying the image to use to create the container.
    - Execute a command on an already running container, with user specifying which command to run and given some means to choose from the running containers. Your program should wait until the command has completed execution before allowing the user to input further commands.
    - List all secrets.
    - Create a secret (use the python "secrets" module to generate the secret data, prompt user for the secret's name).
    - List all images.
    - Show the history of an image, with user specifying the image to use.
    - Remove an image.

2. You must also provide functionality that allows a user to input 3 things: 1) the path to a python program they have written, 2) the version of python (either 2.7 or 3.14) that the program is written in, and 3) an image name.

    - From these inputs, your program should create a new Dockerfile and use it to create a new docker image with the specified image name, which defines a container for the python program.
    - Run a container created from the image above.
    - Print the output from the container.

    *Assume that the user-supplied python program is located in the current directory, and **requires boto3 only if Python 3.14** is being used. Both versions have no other associated libraries/packages necessary to run them.*

3. Your program must provide an option to run a pre-defined scenario: This scenario will use Docker Compose to create some containers which are linked/communicating with one another and **provide some form of output (not necessarily part of the python program) that clearly demonstrates that the link is successful**. *You may not use the sample voting app that was shown in lectures*.

   You can either create the containers yourself (e.g. write some python programs) or take existing ones (provide references if you did use anything pre-existing). You may hardcode in exactly what containers are created by Docker Compose in the scenario (i.e. the user does not provide any input). Attempt to connect at least 3 different types of containers, and not more than 5. The more different container types you have interacting, the better (up to the maximum of 5).

4. Provide a ReadMe pdf document which includes a description of the program, how to use it, and any necessary requirements/details for running the script (e.g. any library dependencies that may need to be installed and which file to run to start the program). The ReadMe should also contain screenshots demonstrating all working functionality as well as a clear listing of incomplete and non-functional sections. For the pre-defined scenario in part (3), you must clearly show evidence of communication having occurred between the containers.

Kubernetes (50%):

1. You will write a python program for interacting with Kubernetes, assuming that you have 1 master server and 2 worker nodes deployed (as shown in the lectures). You may choose how the user interacts with the program, but it must provide the following basic functionality:

   - List all pods.
   - Describe a pod, specified by the user.
   - Create a Deployment with 2-4 pods, specified by the user, from an image for "nginx: 1.28.0-perl" or "mysql:8.4.7", also specified by the user.
   - Scale the number of pods for any deployment created by the previous task, in the range from 1-4 pods per deployment. Deployment and pod count specified by the user.
   - Perform a *Rolling Update* on the pods from a deployment to run a newer version of nginx or mysql. The user should specify which one to upgrade and what tag to upgrade to. You may assume the Deployment being upgraded was created as part of the 3rd task.
   - Provide an option to rollback the update of the previous task, in case the user provided an invalid tag.
   - Create a pod on every worker node (1 per node), type of the pod specified by the user.
   - Delete all pods (created by any of the previous tasks).

2. Your program must provide a function that runs the following pre-defined scenario: Create multiple pods (of at least 2 different types and not more than 5 types, with at least 2 replicas of each type) which communicate via ClusterIPs, and some of which are externally exposed through a LoadBalancer Service. The higher the complexity of the system the better. You must provide references to any external resources used.

3. Provide a ReadMe document which includes a description of the program, how to use it, and any necessary requirements/details for running the script (e.g. any library dependencies that may need to be installed beyond the basic setup for the VMs shown in the lectures, and should specify which file to run to start the program). The ReadMe should also contain screenshots demonstrating all working functionality as well as a clear listing of incomplete and non-functional sections. For the pre-defined scenario, you must clearly show evidence of communication having occurred between the pods.

Importantly, for both Docker and Kubernetes, please note that you should use best practice when writing your code i.e. use classes/methods where sensible, minimise hardcoding paths, and minimising code duplication. Code should be robust and include error checking so it can deal accordingly with error. Marks will be awarded for well written robust code.

Your code must include clear concise comments, for functions and also for any complex parts of code which explain what they are doing. These should clearly demonstrate your understanding of the submitted code, i.e. that you understand what type of objects you're working with, why you're calling a method with certain arguments etc... Marks will be awarded for this.

Submission Details:

Due Date: 7th December 2025, 23:59.

No-Penalty Late Submission until 21st December 2025, 23:59. Note that any assignments submitted after this exact time will receive a score of 0.

Submit your code (all Python scripts and any other files) along with the ReadMe files in a single zipped directory via Canvas. This directory should be named to have your name and student number e.g. "David Stynes R00012345". Marks will be deducted for incorrectly named zip file submissions. An assignment submission facility will be made available in Canvas.

Unlike assignment 1, there is no need to leave any cloud resources active for this assignment.

At the discretion of the examiner, students may be briefly interviewed to demonstrate their understanding of the submitted code.