Project: Containers: Docker and Kubernetes

Due: 02/15/2026 11:59 PM

Automate the build and deployment (Task)

Automate the build and deployment of microservices with CI/CD

In the previous tasks, you gained experience in constructing and launching the given microservices application. Now, it's time to streamline the deployment process through automation. Your task is to set up a CI/CD pipeline using GitHub Actions with four distinct jobs. The pipeline will:

1. Check and identify any updates or modifications in the codebase of each microservice.
2. Rebuild and push updated Docker images to Azure Container Registry (ACR) and Google Artifact Registry (GAR) when changes are detected.
3. Deploy the microservices to Google Kubernetes Engine (GKE) and Azure Kubernetes Service (AKS) clusters under two scenarios:
   Following the successful rebuild and push of Docker images.
   Upon detecting modifications in your Helm chart configurations.

Prior to proceeding with this task, please read and complete the `Intro to GitHub Actions` primer.

GitHub Settings

If you are not familiar with the git commands, please refer to this Microsoft Learn Git and GitHub Basic

Course to learn before starting this task.

Authenticate with GitHub using the command below. Choose `GitHub.com` for the account you want to log into, `SSH` as the preferred protocol, and say `Yes` to generate a new SSH key to add to your GitHub account. You can then use your web browser to authenticate with GitHub CLI.

```
$ gh auth login
```

Navigate to the `task7` directory and initialize it as a Git repository using the following command:

```
$ git init -b main
```

Configure your user information to let GitHub know who's committing the changes.

```
$ git config --global user.email "<your-
$ git config --global user.name "<your-u
```

Copy all of the files related to the microservices to the remote repository using the following commands:

```
$ cp -r ../task5/Ingress ../task5/helm .
$ cp -r ../task5/group-chat-service chat
```

Create a new repository with the name `containers-devops` from the local repository and push all of the existing commits using the following commands:

```
$ git add . && git commit -m "initial co
$ gh repo create containers-devops --sou
```

If you get a `Permission denied (publickey)` error, try following the troubleshooting documentation

or generating and adding a new SSH key on your student VM.

By following these steps, you will have a Git repository set up in your task7 directory with all the necessary files. You can confirm that the new repository has been created by visiting your GitHub repositories page.

Authentication Settings

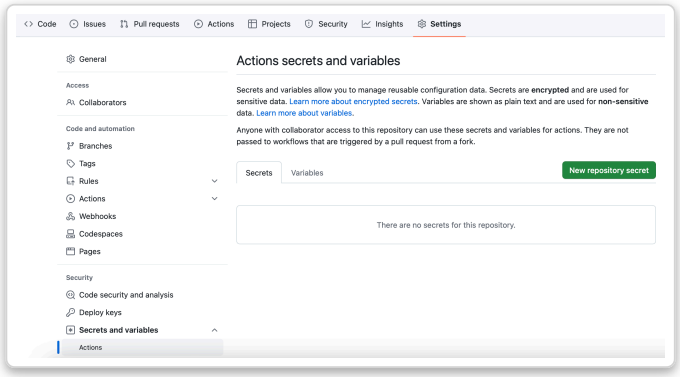
Before proceeding, it's important to set up several GitHub Actions Secrets that contain the necessary resource identifiers and credentials. This will allow GitHub Actions to access your GCP and Azure resources seamlessly.

Here's a list of the tokens that you need to configure in your GitHub Secrets:

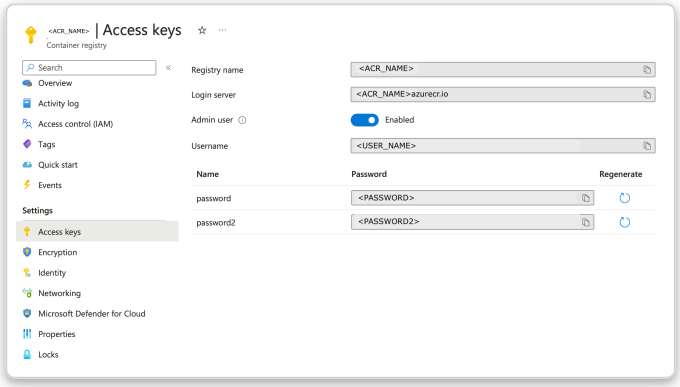
Secret	Description
AZ_CONTAINER_REGISTRY	The login server name of your Azure Container Registry (.azurecr.io)
AZ_REGISTRY_USERNAME	The username for GitHub Actions to log in to your ACR
AZ_REGISTRY_PASSWORD	The password for GitHub Actions to log in to your ACR
AZURE_CLIENT_ID	The client id of a service principal or a user-assigned managed identity
AZURE_SUBSCRIPTION_ID	The subscription ID
AZURE_TENANT_ID	The tenant ID
GCP_SA_KEY	The service account key for GCP

Below are the detailed instructions to set up those tokens:

1. Open your code repository in GitHub and go to the `Settings` page. Under the `Secrets and variables` section, click on `Actions`. Then navigate to the `Secrets` tab and click on `New repository secret` as shown in the image below:



2. Open the Azure Portal in a separate tab and search for "Container registries". On the Container registries page, navigate to the ACR you provisioned for this project. Under "Settings", select "Access keys" and enable the "Admin user". You should then be able to view the "Username" and "Password" as shown below:



3. To add a new secret, click on `New secret` in the GitHub repository settings. For the first secret, specify the `Name` as `AZ_CONTAINER_REGISTRY` and the `Value` as the Login server (<lowercased_acr_name>.azurecr.io). You can alternatively use the GitHub CLI with the command.

```
$ gh secret set AZ_CONTAINER_REGISTRY
```

4. Add the new secret `AZ_REGISTRY_USERNAME` using the `Username` obtained in Step 2 as the value or use the GitHub CLI with the command.

```
$ gh secret set AZ_REGISTRY_USERNAME -
```

5. Add a new secret `AZ_REGISTRY_PASSWORD` using the `Password` obtained in Step 2 as the value or use the GitHub CLI with the command.

```
$ gh secret set AZ_REGISTRY_PASSWORD -
```

6. We will use OpenID Connect within the CI/CD workflows to authenticate with Azure. OpenID Connect (OIDC) allows your GitHub Actions workflows to access resources in Azure, without needing to store the Azure credentials as long-lived GitHub secrets. Using Azure Login Action with OIDC involves the following two steps:
   - Create a user-assigned managed identity and assign a role to it
   - Configure a federated identity credential on a user-assigned managed identity

   Step 1: Create a user-assigned managed identity and assign a role to it

   Use the az identity create command to create a user-assigned managed identity. Specify the `<your-resource-group>` and `<your-identity-name>` parameter values with your own values.

```
export RESOURCE_GROUP="<your-resource-
export USER_ASSIGNED_IDENTITY_NAME="<i
az identity create -g $RESOURCE_GROUP
```

   After creating the identity, retrieve the principal ID of the managed identity by running the following command:

```
az identity show --name $USER_ASSIGNEI
```

Assign the Contributor role to the managed identity. This grants the identity the necessary permissions to perform actions in Azure. Replace with the principal ID obtained from the previous command.

```
export SUBSCRIPTION_ID="<your-subscrip
export PRINCIPAL_ID="<your-principal-i

az role assignment create --assignee $
```

Step 2: Configure a federated identity credential on a user-assigned managed identity

Set the variables and use the command to create the federated identity credential:

```
export GITHUB_USERNAME="<your-github-u
export GITHUB_REPO="containers-devops'
export BRANCH_NAME="main"
export CREDENTIAL_NAME="<federated-cre

az identity federated-credential creat
--identity-name $USER_ASSIGNED_IDENTIT
--resource-group $RESOURCE_GROUP \
--issuer "https://token.actions.github
--subject "repo:${GITHUB_USERNAME}/${G
--audiences "api://AzureADTokenExchang
```

7. After it, create GitHub Action secrets for following values:
   - `AZURE_CLIENT_ID`: the service principal client ID or user-assigned managed identity client ID
   - `AZURE_SUBSCRIPTION_ID`: the subscription ID
   - `AZURE_TENANT_ID`: the tenant ID

The `AZURE_CLIENT_ID` and `AZURE_TENANT_ID` for your managed identity can be obtained by running the following command:

```
az identity list -g $RESOURCE_GROUP
```

8. Refer to steps 1 to 4 in the `Configuring a service account and storing its credentials` section of this tutorial to obtain the service account key for GCP. Please add the additional policy provided below for access to Artifact registry. It should output a `key.json` file in your current directory. Copy and paste the JSON to the new secret `GCP_SA_KEY`. Make sure you DO NOT commit the `key.json` file.

```
gcloud projects add-iam-policy-binding
--member=serviceAccount:$SA_EMAIL \
--role=roles/artifactregistry.admin
```

Ensure that you have set up all 7 secrets as specified above before continuing with the next steps.

Tagging with Git commit hash

In previous tasks, you've learned about using Semantic Versioning for Docker image tags. This task introduces you to a different strategy: tagging Docker images with the **Git commit hash**.

This strategy refers to tagging your docker image with the Git commit hash (for example, `<IMAGE-NAME>:e06d0b4457096f8ce4b6356592f8276f6b918277`) from which they were built. This allows you to link a Docker image directly to the state of the codebase at the time of the image's creation. It's particularly useful for identifying the exact changes included in a specific image.

For this task, you are required to use **Git commit hash** for tagging Docker images. You are also encouraged

to spend time researching more about tagging strategies for Docker image, as there are numerous approaches, each with its unique advantages and drawbacks.

Environment Variable Settings

Here's a list of the environment variables that you need to configure in your `.github/workflows/cicd.yml` file. You should first add the `env` section between the `on` and `job` sections. Then you can define the variables in the `env` section by following this documentation.

Variable	Description
GCP_CLUSTER_NAME	The name of your GCP cluster
GCP_PROJECT_ID	The ID of your project in GCP
GCP_REGION	The zone of your cluster in GCP
AZ_CONTAINER_REGISTRY	The login server name of your Azure Container Registry (ACR)
AZ_CLUSTER_NAME	The name of your Azure cluster
AZ_RESOURCE_GROUP	The resource group of your Azure cluster
DOCKER_TAG	The docker tag you will use for building images. For this task you must use the git commit hash. Refer to the document to learn about how to extract git commit hash from Github Actions.

Create the Workflow

To create the GitHub Actions workflow YAML file on GitHub to define the pipeline, follow the guideline

below. The starter code is provided in the `.github/workflows/cicd.yml file`.

Workflow Trigger

As indicated by the `on` keyword, the workflow will be triggered by any `push` events to the `main` branch. You do not need to modify this part of the code. Learn more about events that trigger workflows from this document.

First Job: Detect file changes and generate outputs for later jobs

The `check-changed-files` job uses the changed-files action to detect any changes in the codebase and generate outputs for subsequent jobs. While there's no need for you to modify the code of this job, it is important to review the document and understand the objectives and functionalities of this job. It's worth noting that the directory names match the names used in previous tasks, such as `profile-service/` or `helm/`. Please ensure that your current codebase strictly follows the naming.

Second Job: Re-build and Push Docker Images to GAR and ACR

The `build-push-image` job rebuilds and pushes the image to GAR and ACR if there are modifications in the Java source code. Your task is to implement the TODOs in the template to complete this job. All necessary steps required to complete the job are in the template. **Note:** Please DO NOT create new steps.

Third Job: Deploy Microservices to GCP

The `deploy-service-to-gcp` job updates the `values-gcp.yaml` files using the `sed` command if there are new docker images pushed to the Artifact Registry. Following this, it applies updates to the GKE cluster using `helm` commands. Your task is to implement the TODOs in the template to complete this job. All necessary steps required to complete the job are in the template. **Note:** Please DO NOT create new steps.

Introduction To The `Sed` Command

The `sed` command, abbreviation for "stream editor," is a powerful utility in Unix-like systems mainly used for manipulating text in data streams and files. It's most commonly used for text substitution of a file. The typical syntax for invoking the sed command is as follows:

```
$ sed OPTIONS... [SCRIPT] [INPUTFILE...]
```

For example, `sed -i 's/hello/world/' file.txt` replaces the first occurrence of `hello` with `world` in each line of file.txt. The `-i` option edits the file in place. The `/` character is the delimiter, commonly a forward slash, but characters like `@` or `|` can also be used.

> Learn more about the `sed` command: https://www.gnu.org/software/sed/manual/sed.html

Hints for completing the bash script in `run`:

> The `sed` command is used in this job to replace the current Docker image tag defined in the `values-gcp.yaml` reference with the new image you built in the second job. If your values files are configured in a different structure, you may need to adjust the `sed` command accordingly.
>
> Investigate the command required to update an already running Helm chart. Is there a command capable of both installing and updating the helm chart?

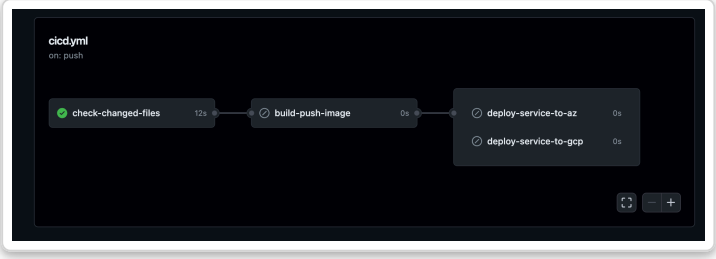
Fourth Job: Deploy Microservices to Azure

The `deploy-service-to-az` job is similar to the previous job, the only difference is we are deploying the microservices to AKS with the `values-azure.yaml` file. Your task is to implement the TODOs in the template to complete this job. All necessary steps required to complete the job are in the template. **Note:** Please DO NOT create new steps.

Trigger the GitHub Actions workflow

Once you have finished the tasks mentioned above, commit your changes of the `.github/workflows/cicd.yml` file and push the commit to the `main` branch to trigger the workflow.

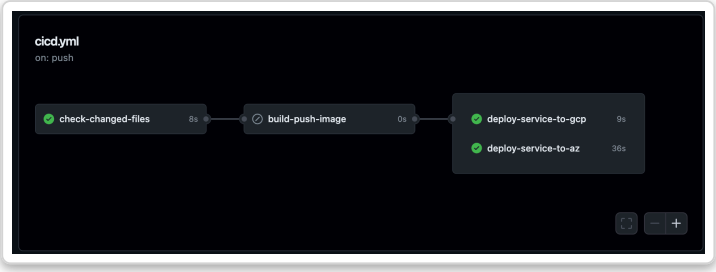
After you push the commit to Github, navigate to the “Actions” tab to review the triggered workflow. Initially, only the first job will be executed to detect changes in the codebase. If you haven’t made any changes to the codebase, the subsequent jobs will not be executed.



In order to trigger the subsequent jobs, you will need to implement changes in your code. For instance, you can update the Helm configurations. This update will then trigger the jobs responsible for installing the modified Helm charts.

1. Navigate to `helm/`
2. Change the number of replicas in the `values.yaml` file in both locations.
3. Commit and push these changes to the main branch.

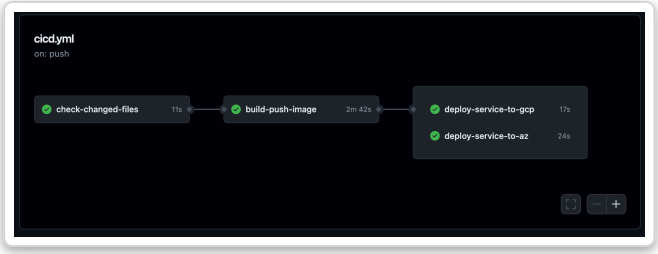
Observing the triggered workflow in “Actions”, you’ll find that the `build-push-image` job are skipped. This is because that there’s no changes to the Java source code or the Dockerfile, thus it’s not necessary to re-build the docker images. However, since we changed the kubernetes configuration, the `deploy-service-to-gcp` and `deploy-service-to-az` jobs are being executed to update the Helm charts on the clusters.



Optional Task

Although it is not required to read and understand the Java source code in this project, you are encouraged to experiment with making some simple changes to the source code under `<service>/src/main/java`. This could be as straightforward as adding a line such as `System.out.println("Cloud Computing is Awesome!");`. These changes will trigger the jobs to rebuild and push Docker Images to the registry. Similarly, you can observe the workflow in the “Actions” tab to verify if it is executed.

Note: To manage artifacts in Google Artifact Registry within your workflow, assign the Artifact Registry Writer role to the service account you created earlier from the IAM role configuration settings.



Wait until the job is successfully completed before validating and submitting your work.

Validate your work

Before you submit, you should check that you have:

1. The latest run of the workflow should have been successfully completed.
2. Appropriate configurations and commands have been added for the tasks in `cicd.yml`.
3. Execute the `kubectl describe deployment` command on your AKS and GKE cluster to verify your changes are reflected.

Edit your information for the submitter

In order to ensure your submissions are successful, it's important to include certain meta information. Specifically, there are three values that must be provided within the `meta.json` file:

```
{
    "github_username": "",
    "repository": "",
    "token": ""
}
```

1. For the `github_username` value, enter the GitHub username that you used for this assignment.
2. For the `repository` value, enter the name of your GitHub repository.
3. For the `token` value, create a new personal access token by following the `Set up Fine-Grained Personal Access Token in GitHub` section in the `Intro to GitHub Actions` primer.

Submissions

The submitter can be found in the `task7` directory. To submit your work, navigate to the root directory and run the command `./submitter`. This will submit your `.github/workflows/cicd.yml` YAML file and the `meta.json` file to the autograder.

Note: In order to obtain full scores, you need to trigger and successfully run all required workflows and then submit.

Troubleshooting

1. If your GitHub Action runs correctly, but you get 0 score on `Build Push Image` task in grading metrics, it probably because you are not using actions we recommend in the `cicd.yaml` file. Please click to those links and carefully these actions.

2. If you encounter any issue during each job (biuld-push-image, deploy-service-to-gcp, and deploy-service-to-az), always go to GitHub Action page to trace back your issues. You may also consider adding `echo` commands in each step to debug your GitHub Action workflow. An Example is shown below

```
# check and print Docker errors
- name: Check Docker build result
  if: steps.docker-build.outcome ==
  run: |
    echo "========== Docker Build Er
    cat login-service/docker-build.l
    echo "==========================
    exit 1
```

← Previous Page Next Page →