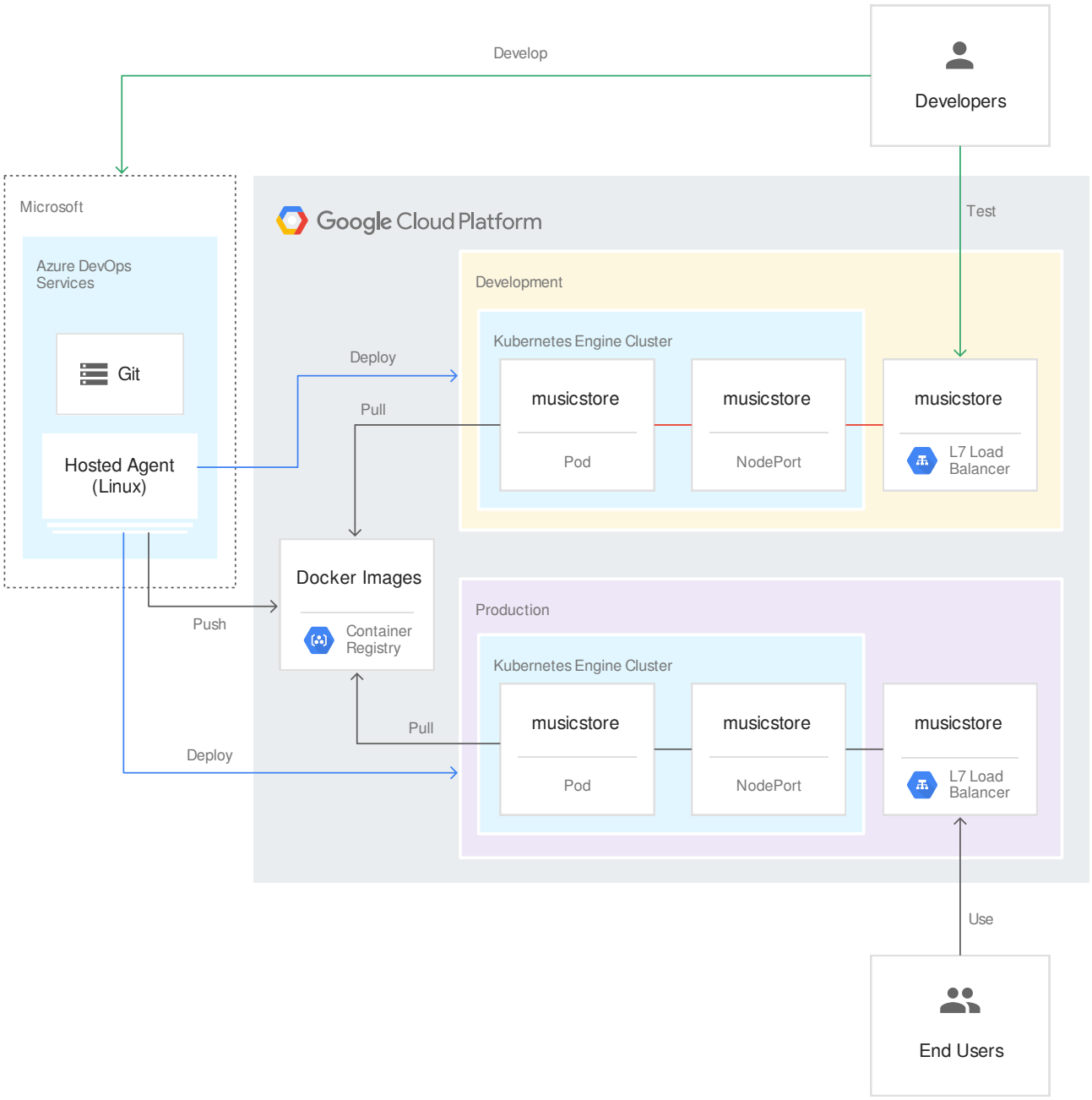
Solutions (https://cloud.google.com/solutions/) Solutions

Creating a CI/CD pipeline with Azure Pipelines and Google Kubernetes Engine

In this tutorial, you learn how to use Azure Pipelines (https://www.visualstudio.com/team-services/) (previously called Visual Studio Team Services), Google Kubernetes Engine (https://cloud.google.com/kubernetes-engine/) (GKE), and Container Registry (https://cloud.google.com/container-registry/) to create a continuous integration/continuous deployment (CI/CD) pipeline. The tutorial uses the ASP.NET MusicStore (https://github.com/aspnet/MusicStore) web application, which is based on ASP.NET Core (https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-2.1).

The CI/CD pipeline uses two separate GKE clusters, one for testing and one for production. At the beginning of the pipeline, developers commit changes to the example codebase. This action triggers the pipeline to create a release and to deploy it to the development cluster. A release manager can then promote the release so that it's deployed into the production cluster. The following diagram illustrates this process.



The tutorial assumes that you have basic knowledge of .NET Core, Azure Pipelines, and GKE. The tutorial also requires you to have administrative access to an Azure DevOps account.

Objectives

- Connect Container Registry to Azure Pipelines for publishing Docker images.

- Prepare a .NET Core sample application (https://github.com/aspnet/MusicStore) for deployment into GKE.
- Authenticate securely against GKE without having to use legacy authentication.
- Use Azure Pipelines release management to orchestrate GKE deployments.

Costs

This tutorial uses billable components of Google Cloud, including:

- GKE (https://cloud.google.com/kubernetes-engine/pricing)
- Cloud Load Balancing (https://cloud.google.com/compute/network-pricing#lb)
- Cloud Storage (https://cloud.google.com/storage/pricing) (for Container Registry)

Use the Pricing Calculator (https://cloud.google.com/products/calculator/) to generate a cost estimate based on your projected usage. Check the Azure DevOps pricing page (https://azure.microsoft.com/en-us/pricing/details/devops/azure-devops-services/) for any fees that might apply for using Azure DevOps.

Before you begin

It's usually advisable to use separate projects for development and production workloads so that identity and access management (IAM) roles and permissions can be granted individually. For the sake of simplicity, this tutorial uses a single project for both GKE clusters, one for development and one for production.

1. In the Cloud Console, on the project selector page, select or create a Cloud project.

   Note: If you don't plan to keep the resources that you create in this procedure, create a project instead of selecting an existing project. After you finish these steps, you can delete the project, removing all resources associated with the project.

   GO TO THE PROJECT SELECTOR PAGE (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/PROJECTSELECT

2. Make sure that billing is enabled for your Google Cloud project. Learn how to confirm billing is enabled for your project (https://cloud.google.com/billing/docs/how-to/modify-project).
3. Make sure you have an Azure DevOps account and have administrator access to it. If you don't yet have an Azure DevOps account, you can sign up on the Azure DevOps home page (https://go.microsoft.com/fwlink/?LinkId=307137).

Note: You can either use a command-line Git client or Visual Studio to follow this tutorial. If you use Visual Studio, make sure that it's connected to your Azure DevOps account.

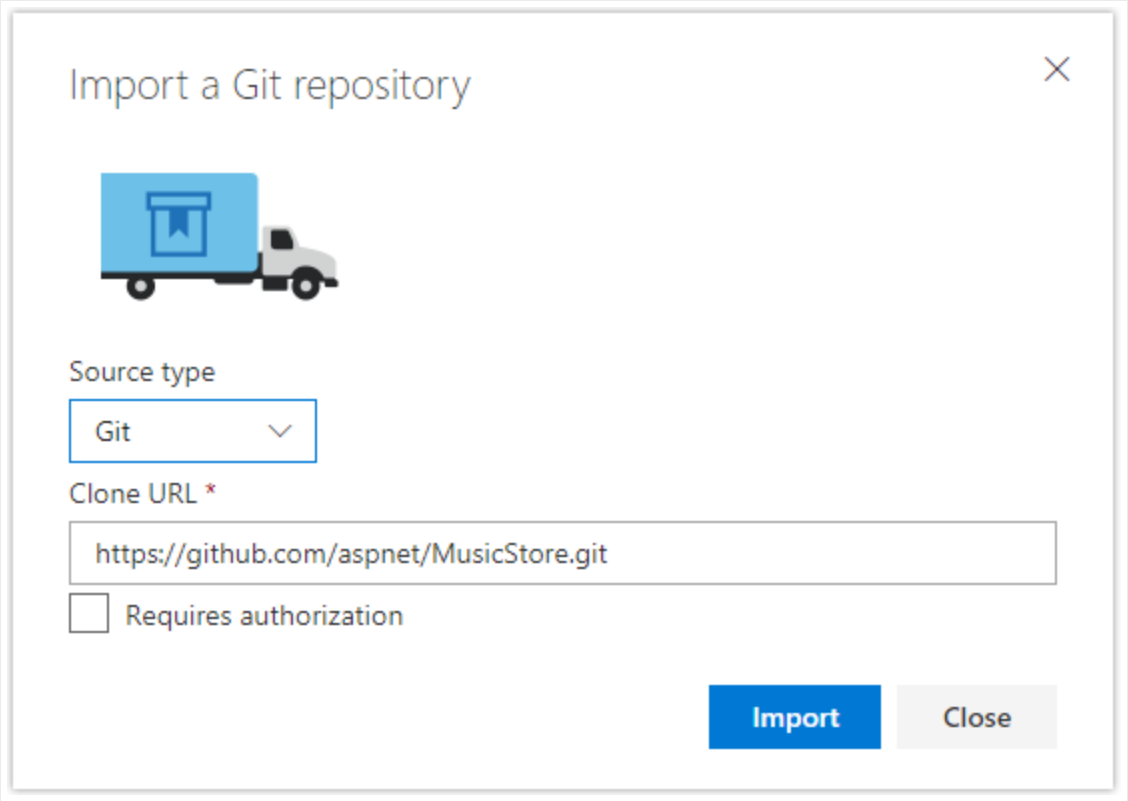
Creating an Azure DevOps project

You use Azure DevOps to manage the source code, run builds and tests, and orchestrate the deployment to GKE. To begin, you create a new project in your Azure DevOps account.

1. Go to the Azure DevOps home page (`https://dev.azure.com/[YOUR_AZURE_DEVOPS_ACCOUNT_NAME]`).
2. Click **Create Project**.
3. Enter a project name, such as `Music Store`.
4. Set **Visibility** to **Private**, and then click **Create project**.
5. After the project has been created, in the menu on the left, click **Repos**.
6. Click **Import** to fork the Music Store repository from GitHub. Set the following values:
    - **Source type**: `Git`
    - **Clone URL**:

      ```
      https://github.com/aspnet/MusicStore.git
      ```

    - Leave the **Requires authorization** checkbox unselected.



7. Click **Import**.

   When the import process is done, you see the source code of the MusicStore application.

Building continuously

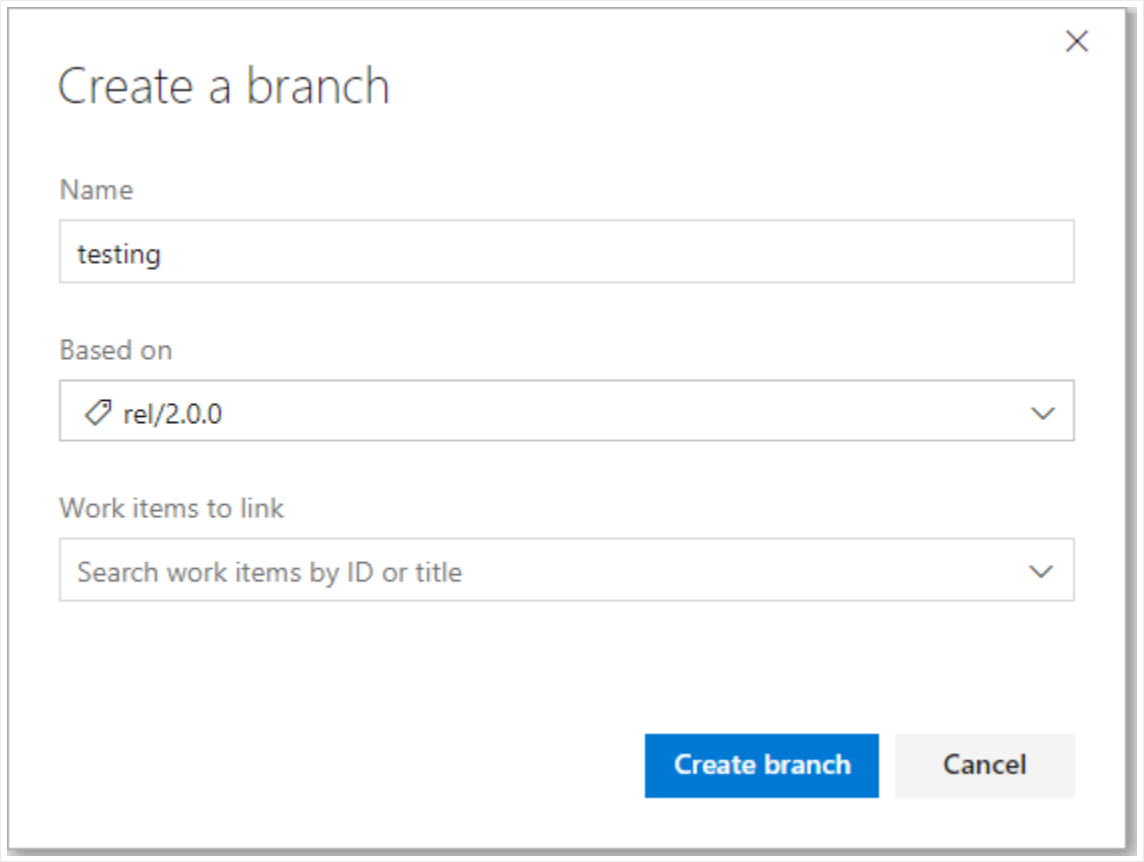
You can now use Azure Pipelines to set up continuous integration. For each commit that's pushed to the Git repository, Azure Pipelines builds the code and packages the build artifacts into a Docker container. The container is then published to Container Registry.

Creating a testing branch

This tutorial was built and tested with version 2.0.0 of the Music Store application. To make sure that you are using the same version, create a new branch based on the `rel/2.0.0` tag:

1. In the Azure DevOps menu, select **Repos** > **Tags**.
2. In the list of tags, expand **rel** and then right-click the icon next to **2.0.0**.
3. Select **New branch**.

4. Enter `testing` as the branch name and then click **Create branch** to confirm your selections.



By default, Azure Pipelines expects your code to reside in the `master` branch. In order to have it use the `testing` branch, you need to change the default branch.

1. In the Azure DevOps menu, select **Project settings**.
2. Select **Repos** > **Repositories**.
3. In the list of repositories, select the Git repository you imported previously. It should have the same name as your Azure DevOps project.
4. Expand the list of branches by clicking on the arrow next to **Branches**.
5. Select the **testing** branch.

   A **...** button appears next to the name of the branch.
6. Click **...** and select **Set as default branch**.

Building the code

After you create the branch, you can begin to automate the build. Because MusicStore is an ASP.NET Core application, building includes four steps:

- Downloading and installing dependencies.
- Compiling the code.
- Running unit tests.
- Publishing the build results.

Later you'll add additional steps to deploy to GKE. Because GKE is a Linux-based environment, you'll set up the entire build process to run on Linux-based build agents.

Creating a build pipeline

Define a new build pipeline by adding a YAML file to the Git repository:

1. Clone your new Git repository (https://docs.microsoft.com/en-us/azure/devops/repos/git/clone?view=azure-devops&tabs=visual-studio)
   by using Visual Studio or a command-line Git client.
2. In the root of the Git workspace, create a new file named **`azure-pipelines.yml`**.
3. Copy the following code and paste into the file:

```
resources:
- repo: self
  fetchDepth: 1
queue:
  name: Hosted Ubuntu 1604
trigger:
- testing
variables:
  TargetFramework: 'netcoreapp2.0'
  RestoreBuildProjects: 'samples/**/*.csproj'
  TestProjects: 'test/MusicStore.Test/*.csproj'
  BuildConfiguration: 'Release'
  DockerImageName: '[PROJECT-ID]/musicstore'
steps:
- task: DotNetCoreCLI@2
  displayName: Restore
  inputs:
    command: restore
```

```
    projects: '$(RestoreBuildProjects)'
    feedsToUse: config
    nugetConfigPath: NuGet.config
- task: DotNetCoreCLI@2
  displayName: Build
  inputs:
    projects: '$(RestoreBuildProjects)'
    arguments: '--configuration $(BuildConfiguration) --framework=$(TargetFrame
- task: DotNetCoreCLI@2
  displayName: Test
  inputs:
    command: test
    projects: '$(TestProjects)'
    arguments: '--configuration $(BuildConfiguration) --framework=$(TargetFrame
- task: DotNetCoreCLI@2
  displayName: Publish
  inputs:
    command: publish
    publishWebProjects: True
    arguments: '--configuration $(BuildConfiguration) --framework=$(TargetFrame
    zipAfterPublish: false
    modifyOutputPath: false
```

4. In the `variables` section, replace `[PROJECT_ID]` with the name of your Google Cloud project, then save the file.
5. Commit your changes and push them to Azure Pipelines:

VISUAL STUDIO | COMMAND LINE

a. Open Team Explorer and click the **Home** icon at upper left to switch to the **Home** view.

b. Click **Changes**.

c. Enter a commit message like `Add build definition`.

d. Click **Commit All and Push**.

6. In the Azure DevOps menu, select **Pipelines** > **Builds**.

   Observe that a build definition has been created based on the YAML file that you have committed to the Git repository.

Publishing Docker images

To deploy the MusicStore application to GKE, the application must be packaged as a Docker container and published to Container Registry. You will now extend the build definition to automate these steps.

Setting up a service account for publishing images

Connecting to Container Registry requires that Azure Pipelines can authenticate with GCP. To do this, create a service account in GCP that's dedicated to this purpose.

1. Switch to your project in the Cloud Console and open Cloud Shell.

   OPEN CLOUD SHELL (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/?CLOUDSHELL=TRUE)

2. To save time typing your project ID and Compute Engine zone options, set default configuration values by running the following commands:

   ```
   gcloud config set project [PROJECT_ID]
   gcloud config set compute/zone [ZONE]
   ```

   Replace `[PROJECT_ID]` with the ID of your GCP project and replace `[ZONE]` with the name of the zone that you're going to use for creating resources. If you're unsure about which zone to pick, use `us-central1-a`.

   Example:

   ```
   gcloud config set project azure-pipelines-test-project-12345
   gcloud config set compute/zone us-central1-a
   ```

3. Enable the Container Registry API for your project:

   ```
   gcloud services enable containerregistry.googleapis.com
   ```

4. Create a service account for Azure Pipelines to publish Docker images:

   ```
   gcloud iam service-accounts create azure-pipelines-publisher --display-name 'Az
   ```

5. Assign the Storage Admin IAM role (https://cloud.google.com/storage/docs/access-control/iam-roles) to the service account:

   ```
   PROJECT_NUMBER=$(gcloud projects describe \
       $(gcloud config get-value core/project) \
   ```

```
    --format='value(projectNumber)')

AZURE_PIPELINES_PUBLISHER=$(gcloud iam service-accounts list \
    --filter="displayName:Azure Pipelines Publisher" \
    --format='value(email)')

gcloud projects add-iam-policy-binding \
    $(gcloud config get-value core/project) \
    --member serviceAccount:$AZURE_PIPELINES_PUBLISHER \
    --role roles/storage.admin
```

6. Generate a service account key:

```
gcloud iam service-accounts keys create \
    azure-pipelines-publisher.json --iam-account $AZURE_PIPELINES_PUBLISHER

tr -d '\n' < azure-pipelines-publisher.json > azure-pipelines-publisher-oneline
```

7. Launch Code Editor by clicking the button in the upper-right corner of Cloud Shell:

8. Open the file named `azure-pipelines-publisher-oneline.json`. You'll need the content of this file in one of the following steps.

Connecting Azure Pipelines to Container Registry

With the service account created, you can now connect Azure Pipelines to Container Registry.

1. In the Azure DevOps menu, select **Project settings** and then select **Pipelines** > **Service connections**.
2. Click **Create service connection**.
3. From the list, select **Docker Registry** and click **Next**.
4. In the dialog, enter values for the following fields:
    - **Registry type**: **Others**
    - **Docker Registry**: `https://gcr.io/[PROJECT-ID]`, where `[PROJECT-ID]` is the name of your Google Cloud project.

      Example: `https://gcr.io/azure-pipelines-test-project-12345`
    - **Docker ID**: `_json_key`

- **Docker Password**: Paste the content of `azure-pipelines-publisher-oneline.json`
- **Service connection name**: `gcr-tutorial`

5. Click **Save** to create the connection.

Creating a Dockerfile

1. In the root of your Git workspace, create a new file named `Dockerfile`.
2. Copy the following code and paste into the file, and then save the file:

```
FROM microsoft/aspnetcore:2.0.0
WORKDIR /app
COPY samples/MusicStore/bin/Release/netcoreapp2.0/publish /app/
ENTRYPOINT ["dotnet", "MusicStore.dll"]
```

3. Create a another file named `deployment.yaml` in the root of your Git workspace. Leave the file empty for now.
4. Commit your changes:

VISUAL STUDIO | COMMAND LINE

a. Open Team Explorer and click the **Home** icon at upper left to switch to the **Home** view.

b. Click **Changes**.

c. Enter a commit message like `Add Dockerfile and placeholder for the Kubernetes manifest`.

d. Click **Commit All**.

Extending the build definition to build a Docker image

With all of the necessary files checked in, you can now extend the build definition.

1. Open the file `azure-pipelines.yml`.
2. Extend the build definition by appending the following piece of code to the file:

```
- task: CmdLine@1
  displayName: 'Lock image version in deployment.yaml'
  inputs:
```

```
    filename: /bin/bash
    arguments: '-c "awk ''{gsub(\"MUSICSTORE_IMAGE\", \"gcr.io/$(DockerImageNam
- task: PublishBuildArtifacts@1
  displayName: 'Publish Artifact'
  inputs:
    PathtoPublish: '$(build.artifactstagingdirectory)'
- task: Docker@0
  displayName: 'Build image'
  inputs:
    containerregistrytype: 'Container Registry'
    dockerRegistryConnection: 'gcr-tutorial'
    imageName: '$(DockerImageName):$(Build.BuildId)'
- task: Docker@0
  displayName: 'Publish image'
  inputs:
    containerregistrytype: 'Container Registry'
    dockerRegistryConnection: 'gcr-tutorial'
    action: 'Push an image'
    imageName: '$(DockerImageName):$(Build.BuildId)'
```

3. Commit your changes and push them to Azure Pipelines:

VISUAL STUDIO | COMMAND LINE

a. Open Team Explorer and click the **Home** icon at upper left to switch to the **Home** view.

b. Click **Changes**.

c. Enter a commit message like `Extend build definition to build Docker image`.

d. Click **Commit All and Push**.

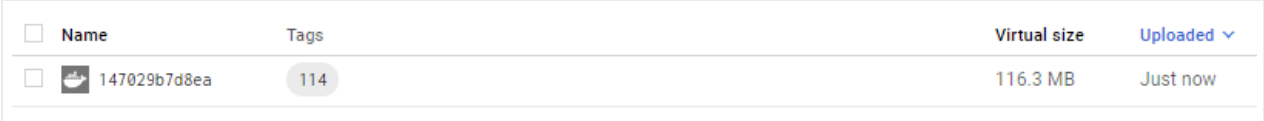
4. In the Azure DevOps menu, select **Pipelines** > **Builds**.

   Observe that a new build has automatically been triggered. It might take around 2 minutes for the build to complete.

   If the build fails with the error message `Step input dockerRegistryConnection references service connection gcr-tutorial which could not be found`, you might need to re-save your pipeline (https://docs.microsoft.com/en-us/azure/devops/pipelines/process/resources?view=vsts#troubleshooting-authorization-for-a-yaml-pipeline).

5. To verify that the image has been published to Container Registry, switch to the Cloud Console, select **Container Registry** > **Images**, and then click **musicstore**.

   Observe that there is a single image, and that the tag of this image corresponds to the numeric ID of the build that was run in Azure Pipelines.



Deploying continuously

With Azure Pipelines automatically building your code and publishing Docker images for each commit, you can now turn your attention to deployment.

Unlike some other continuous integration systems, Azure Pipelines makes a distinction between *building* and *deploying*, and it provides a specialized set of tools labeled Release Management for all of the deployment-related tasks.

Azure Pipelines Release Management is built around these concepts:

- A *release* refers to a set of artifacts that make up a specific version of your application and that are usually the result of a build process.
- *Deployment* refers to the process of taking a release and deploying it into a specific environment.
- A deployment performs a set of *tasks*, which can be grouped in *jobs*.
- *Stages* allow you to segment your pipeline and can be used to orchestrate deployments to multiple environments, for example development and testing environments.

The primary artifact that the MusicStore build process produces is the Docker image. However, because the Docker image is published to Container Registry, the image is outside the scope of Azure Pipelines. The image therefore doesn't serve well as the definition of a release.

To deploy to Kubernetes, you also need a *manifest*, which resembles a bill of materials. The manifest not only defines the resources that Kubernetes is supposed to create and manage, but also specifies the exact version of the Docker image to use. The Kubernetes manifest is well suited to serve as the artifact that defines the release in Azure Pipelines Release Management.

Configuring the Kubernetes deployment

To run the MusicStore in Kubernetes, you need the following resources:

- A *Deployment* that defines a single pod that runs the Docker image produced by the build.
- A *NodePort* service that makes the pod accessible to a load balancer.
- An *Ingress* that exposes the application to the public internet by using a Cloud HTTP(S) load balancer (https://cloud.google.com/compute/docs/load-balancing/http/).

With the MusicStore application, you can use either SQL Server or an embedded, locally stored database. For the sake of simplicity, use the default configuration that relies on the embedded database, although it comes with two restrictions:

- Only a single copy of the pod can run at a time. Otherwise users might see different data depending on which pod serves them.
- Any data changes are lost whenever the pod is restarted, unless you change the deployment to use persistent volumes (https://cloud.google.com/kubernetes-engine/docs/how-to/stateful-apps). (We do not cover this scenario in the tutorial.)

To define these Kubernetes resources, you perform the following steps:

1. Open `deployment.yaml` and paste in the following code, and then save the file:

```
apiVersion: v1
kind: Service
metadata:
  name: musicstore
spec:
  ports:
  - port: 80
    targetPort: 80
    protocol: TCP
    name: http
  selector:
    app: musicstore
  type: NodePort

---
apiVersion: extensions/v1beta1
kind: Ingress
```

```
metadata:
  name: musicstore
spec:
  backend:
    serviceName: musicstore
    servicePort: 80

---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: musicstore
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: musicstore
    spec:
      containers:
      - name: musicstore
        image: MUSICSTORE_IMAGE
        ports:
          - containerPort: 80
        livenessProbe:      # Used by deployment controller
          httpGet:
            path: /
            port: 80
          initialDelaySeconds: 5
          periodSeconds: 5
        readinessProbe:     # Used by Ingress/GCLB
          httpGet:
            path: /
            port: 80
          initialDelaySeconds: 3
          periodSeconds: 5
        resources:
          limits:
            memory: 1024Mi
          requests:
            memory: 768Mi
```

2. Commit your changes and push them to Azure Pipelines:

VISUAL STUDIO COMMAND LINE

a. Open Team Explorer and click the **Home** icon at upper left to switch to the **Home** view.

b. Click **Changes**.

c. Enter a commit message like `Add Kubernetes manifest`.

d. Click **Commit All and Push**.

Setting up the development and production environments

Before returning to Azure Pipelines Release Management, you need to create the GKE clusters.

Creating GKE clusters

1. In GCP, open a Cloud Shell instance.
2. To save time typing your project ID and Compute Engine zone options, set the default configuration values by running the following commands:

```
gcloud config set project [PROJECT_ID]
gcloud config set compute/zone [ZONE]
```

Example:

```
gcloud config set project azure-pipelines-test-project-12345
gcloud config set compute/zone us-central1-a
```

3. Enable the GKE API for your project:

```
gcloud services enable container.googleapis.com
```

4. Create the development cluster by using the following command. Note that it might take a few minutes to complete:

```
gcloud container clusters create azure-pipelines-cicd-dev
```

5. Create the production cluster by using the following command. Note that it might take a few minutes to complete:

```
gcloud container clusters create azure-pipelines-cicd-prod
```

Connecting Azure Pipelines to the development cluster

Just as you can use Azure Pipelines to connect to an external Docker registry like Container Registry, Azure Pipelines supports integrating external Kubernetes clusters.

It is possible to authenticate to Container Registry using a Google Cloud service account, but using Google Cloud service accounts is not supported by Azure Pipelines for authenticating with GKE. Instead, you have to use a Kubernetes service account (https://kubernetes.io/docs/reference/access-authn-authz/authentication/#service-account-tokens).

To connect Azure Pipelines to your development cluster, you therefore have to create a Kubernetes service account first.

1. In Cloud Shell, connect to the development cluster:

   ```
   gcloud container clusters get-credentials azure-pipelines-cicd-dev
   ```

2. Create a Kubernetes service account for Azure Pipelines:

   ```
   kubectl create serviceaccount azure-pipelines-deploy
   ```

3. Assign the `cluster-admin` role to the service account by creating a cluster role binding:

   ```
   kubectl create clusterrolebinding azure-pipelines-deploy --clusterrole=cluster-
   ```

4. Determine the IP address of the cluster:

   ```
   gcloud container clusters describe azure-pipelines-cicd-dev --format=value\(end
   ```

   You will need this address in a moment.

5. In the Azure DevOps menu, select **Project settings** and then select **Pipelines** > **Service connections**.
6. Click **New service connection**.
7. Select **Kubernetes** and click **Next**.
8. Configure the following settings.

> **Note:** To get some of these values, you must run `kubectl` commands in Cloud Shell and then copy the values into the Azure page.

- **Authentication method**: **Service account**.
- **Server URL**: `https://[MASTER-IP]/`. Replace `[MASTER-IP]` with the IP address that you determined earlier.
- **Secret**: Run the following command in Cloud Shell and copy the output:

```
kubectl get secret $(kubectl get serviceaccounts azure-pipelines-deploy -o
```

- **Service connection name**: `azure-pipelines-cicd-dev`.

9. Click **Save**.

Connecting Azure Pipelines to the production cluster

To connect Azure Pipelines to your production cluster, you can follow the same approach.

1. In Cloud Shell, connect to the production cluster:

```
gcloud container clusters get-credentials azure-pipelines-cicd-prod
```

2. Create a Kubernetes service account for Azure Pipelines:

```
kubectl create serviceaccount azure-pipelines-deploy
```

3. Assign the `cluster-admin` role to the service account by creating a cluster role binding:

```
kubectl create clusterrolebinding azure-pipelines-deploy --clusterrole=cluster-
```

4. Determine the IP address of the cluster:

```
gcloud container clusters describe azure-pipelines-cicd-prod --format=value\(en
```

   You will need this address in a moment.

5. In the Azure DevOps menu, select **Project settings** and then select **Pipelines** > **Service connections**.

6. Click **New service connection**.
7. Select **Kubernetes** and click **Next**.
8. Configure the following settings:

   Note: To get some of these values, you must run `kubectl` commands in Cloud Shell and then copy the values into the Azure page.

   - **Authentication method**: **Service account**.
   - **Server URL**: `https://[MASTER-IP]/`. Replace `[MASTER-IP]` with the IP address that you determined earlier.
   - **Secret**: Run the following command in Cloud Shell and copy the output:

     ```
     kubectl get secret $(kubectl get serviceaccounts azure-pipelines-deploy -o
     ```

   - **Service connection name**: `azure-pipelines-cicd-prod`.
9. Click **Save**.

Configuring the release pipeline

After you set up the GKE infrastructure, you return to Azure Pipelines to automate the deployment, which includes the following:

- Deploying to the development environment.
- Requesting manual approval before initiating a deployment to the production environment.
- Deploying to the production environment.

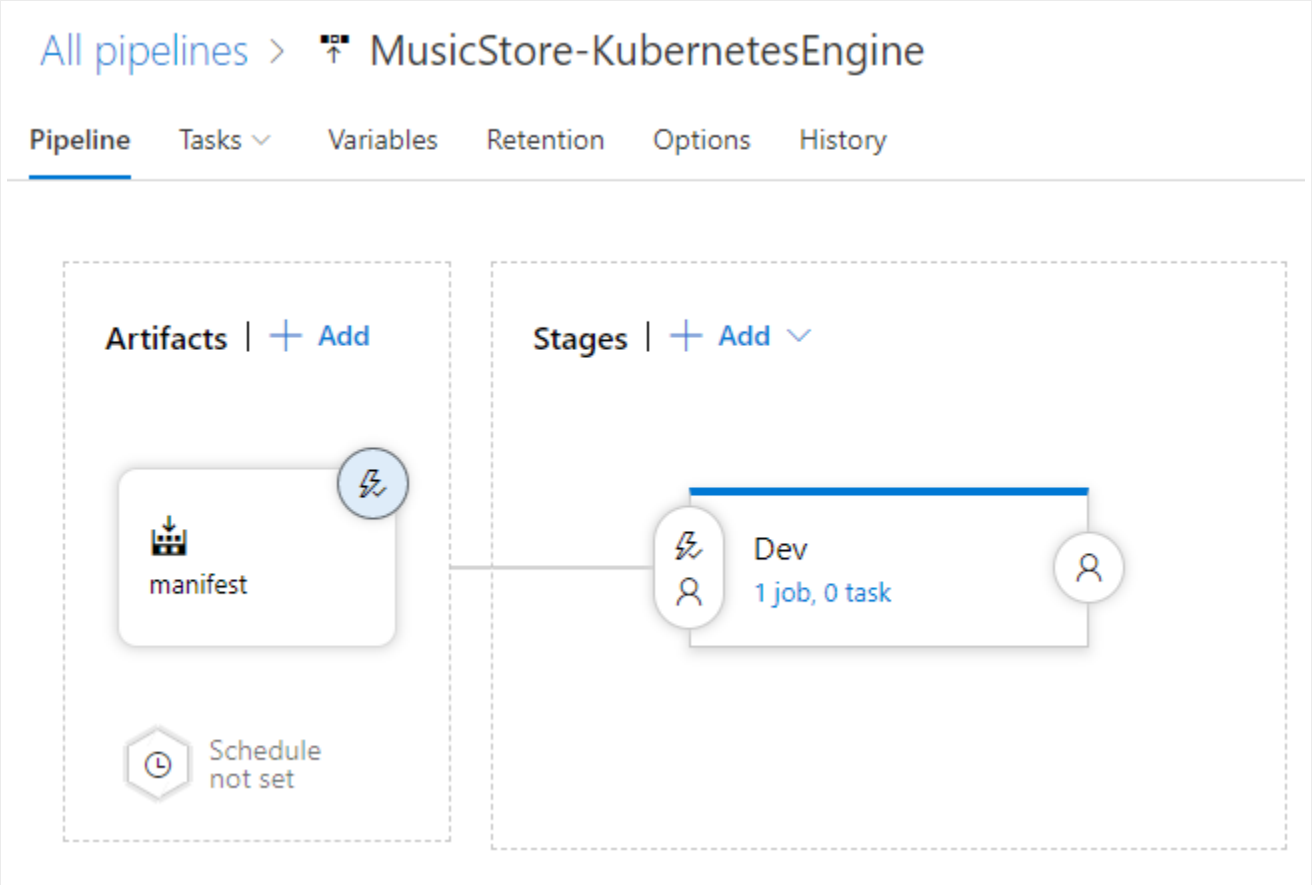
Creating a release definition

As a first step, create a new release definition.

1. In the Azure DevOps menu, select **Pipelines** > **Releases**.
2. Click **New pipeline**.
3. From the list of templates, select **Empty job**.

4. When you're prompted for a name for the stage, enter `Dev`.
5. At the top of the screen, name the release **MusicStore-KubernetesEngine**.
6. In the pipeline diagram, next to **Artifacts**, click **Add**.
7. Select **Build** and add the following settings:
    - **Source (build pipeline)**: Select the build definition (there should be only one option)
    - **Default version**: `Latest`
    - **Source Alias:** `manifest`
8. Click **Add**.
9. On the **Artifact** box, click the lightning bolt icon to add a deployment trigger.
10. Under **Continuous deployment trigger**, set the switch to **Enabled**.
11. Click **Save**.
12. Enter a comment if you want, and then confirm by clicking **Save**.

The pipeline now looks like this:



Deploying to the development cluster

With the release definition created, you can now configure the deployment to the GKE development cluster.

1. In the pipeline menu, switch to the **Tasks** tab.
2. Click **Agent job**.
3. Set **Agent specification** to **ubuntu-16.04**.
4. Next to **Agent job**, click the **+** icon to add a step to the phase.
5. Select the **Deploy to Kubernetes** task and click **Add**.
6. Click the newly added task and configure the following settings:
    - **Display name**: `Deploy`
    - **Action**: **deploy**
    - **Kubernetes service connection**: **azure-pipelines-cicd-dev**
    - **Namespace**: `default`
    - **Strategy**: **None**
    - **Manifests**: `manifest/drop/deployment.yaml`
7. Click **Save**.
8. Enter a comment if you want, and then confirm by clicking **OK**.

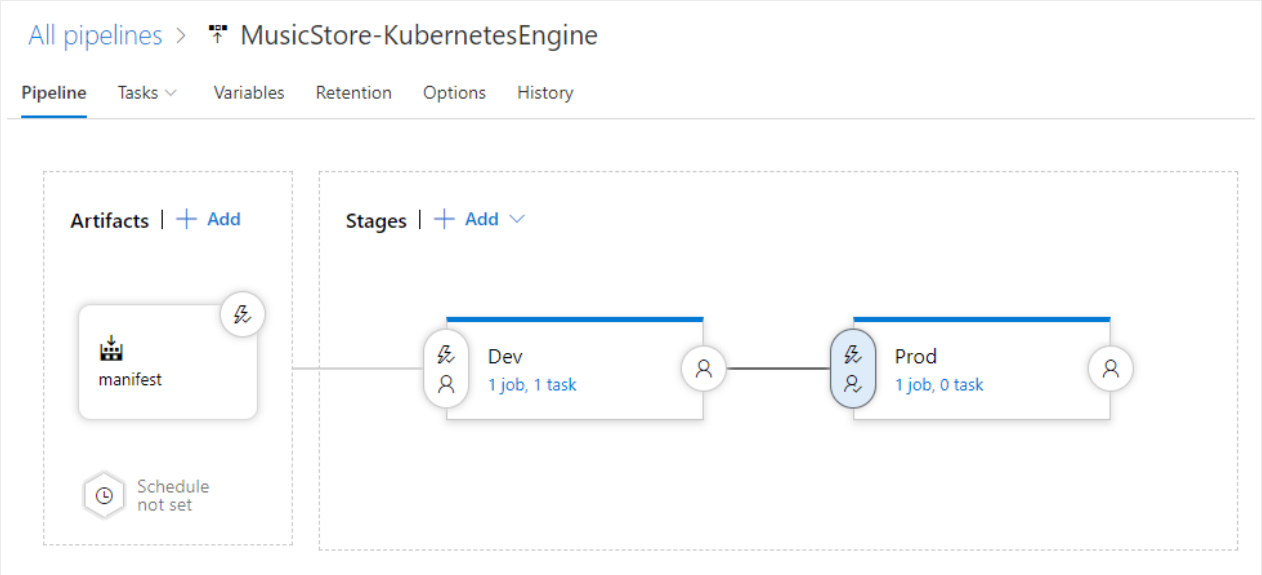
Deploying to the production cluster

Finally, you configure the deployment to the GKE production cluster.

1. In the menu, switch to the **Pipeline** tab.
2. In the **Stages** box, select **Add** > **New stage**.
3. From the list of templates, select **Empty job**.
4. When you're prompted for a name for the stage, enter `Prod`.
5. Click the lightning bolt icon of the newly created stage.
6. Configure the following settings:
    - **Select trigger**: **After stage**
    - **Stages**: **Dev**

- **Pre-deployment approvals**: (enabled)
- **Approvers**: Select your own user name.

The pipeline now looks like this:



7. Switch to the **Tasks** tab.
8. Hold the mouse over the **Tasks** tab and select **Tasks** > **Prod**.
9. Click **Agent job**.
10. Set **Agent specification** to **ubuntu-16.04**.
11. Click the **+** icon to add a step to the phase.
12. Select the **Deploy to Kubernetes** task and click **Add**.
13. Click the newly added task and configure the following settings:
    - **Display name**: `Deploy`
    - **Action**: **deploy**
    - **Kubernetes service connection**: **azure-pipelines-cicd-prod**
    - **Namespace**: `default`
    - **Strategy**: **None**
    - **Manifests**: `manifest/drop/deployment.yaml`
14. Click **Save**.
15. Enter a comment if you want, and then confirm by clicking **OK**.

Running the pipeline

Now that you've configured the entire pipeline, it's time to test it by performing a source code change.

1. In your Git workspace, open the file `samples\MusicStore\config.json`.
2. In line 3, change the `SiteTitle` setting to `ASP.NET MVC Music Store running on Google Kubernetes Engine`.
3. Commit your changes and push them to Azure Pipelines:

VISUAL STUDIO | COMMAND LINE

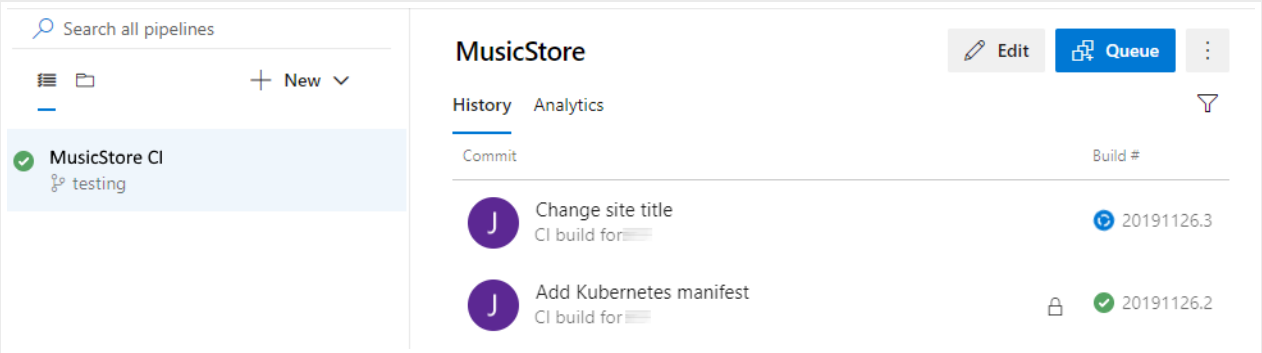
a. Open Team Explorer and click the **Home** icon at upper left to switch to the **Home** view.

b. Click **Changes**.

c. Enter a commit message like `Change site title`.

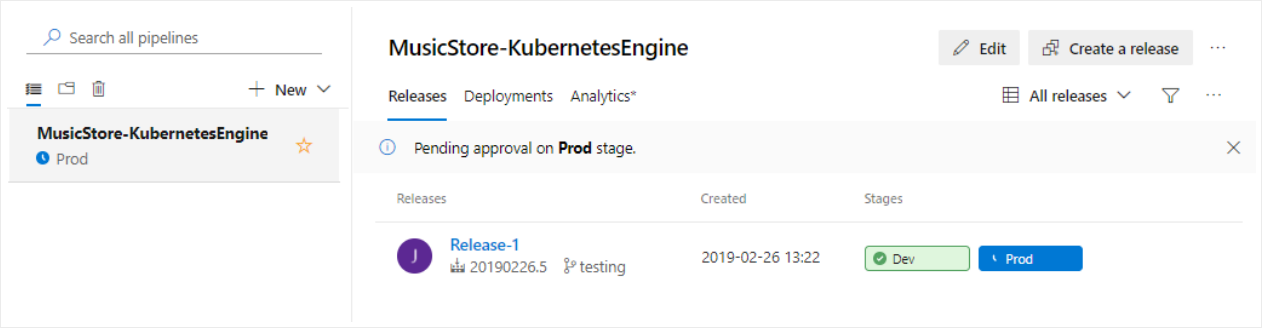
d. Click **Commit All and Push**.

4. In the Azure DevOps menu, select **Pipelines** > **Builds** and observe that a build has been triggered automatically:

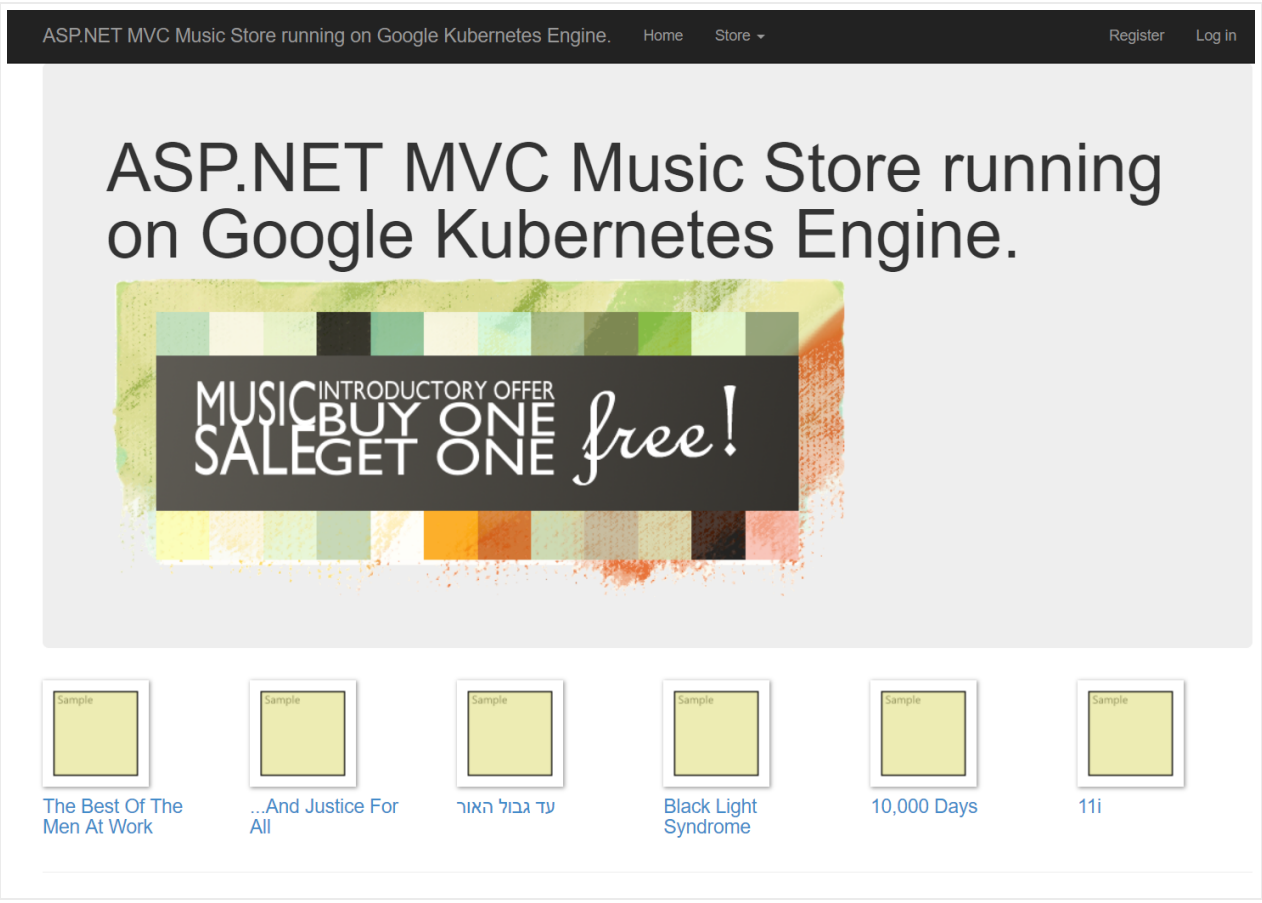


It might take around 2 minutes before the status switches to **Succeeded**.

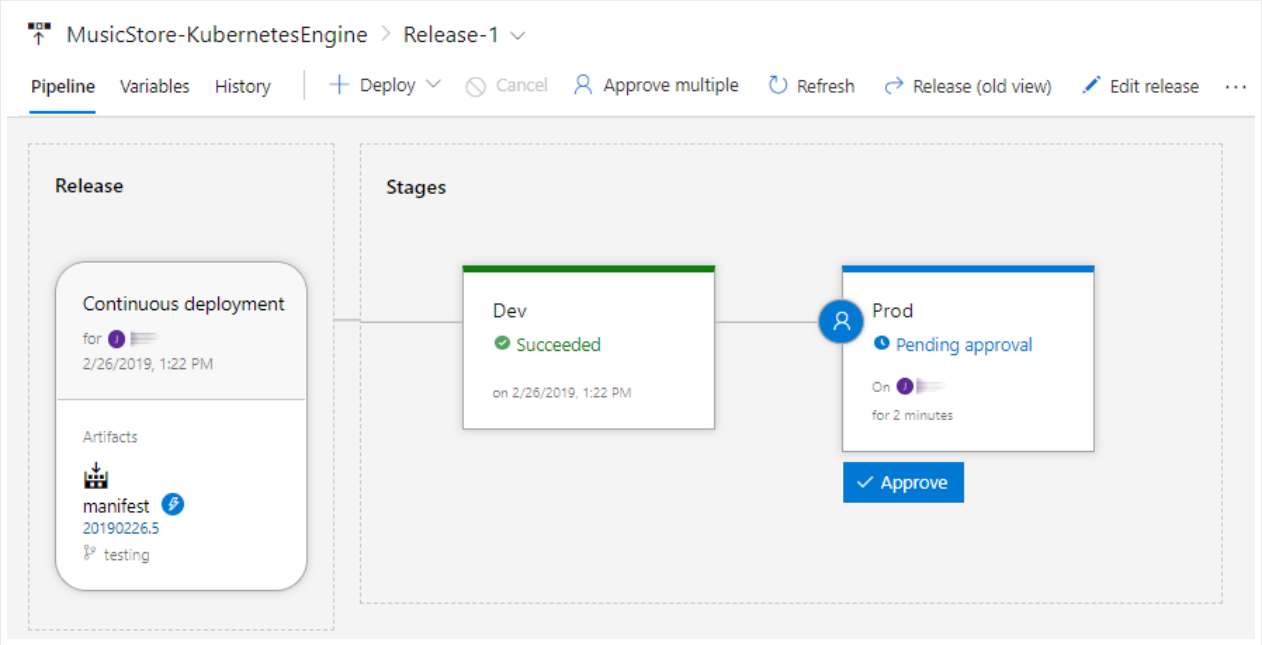
5. When the build is finished, select **Pipelines** > **Releases** and observe that a release process has been initiated:



6. Click **Release-1** to open the details page, and wait for the status of the **Dev** stage to switch to **Succeeded**. You might need to refresh the status by clicking the **Refresh** button in the menu or by reloading the browser page.
7. In the Cloud Console, select **Kubernetes Engine** > **Services & Ingress**.
8. Locate the Ingress service for the **azure-pipelines-cicd-dev** cluster, and wait for its status to switch to **Ok**. This might take several minutes.
9. Open the link in the **Endpoints** column of the same row. You might see an error at first because the load balancer takes a few minutes to become available. When it's ready, observe that the Music Store has been deployed and is using the custom title:



10. In Azure Pipelines, click the **Approve** button located under the **Prod** stage to promote the deployment to the production environment:



    If you don't see the button, you might need to first approve or reject a previous release.

11. Enter a comment if you want, and then confirm by clicking **Approve**.
12. Wait for the status of the **Prod** environment to switch to **Succeeded**. You might need to manually refresh the page in your browser.
13. In the Cloud Console, refresh the **Services** page.
14. Locate the Ingress service for the **azure-pipelines-cicd-prod** cluster and wait for its status to switch to **Ok**. This might take several minutes.
15. Open the link in the **Endpoints** column of the same row. Again, you might see an error at first because the load balancer take a few minutes to become available. When it's ready, you see the MusicStore app with the custom title again, this time running in the production cluster.

Cleaning up

To avoid incurring further costs after you have completed this tutorial, delete the entities that you've created.

Delete the Azure Pipelines project

Delete the project (https://docs.microsoft.com/en-us/vsts/accounts/delete-team-project?view=vsts) in Azure Pipelines. Note that this also causes all source code changes to be lost.

Delete the GCP project

Caution: Deleting a project has the following effects:

- **Everything in the project is deleted.** If you used an existing project for this tutorial, when you delete it, you also delete any other work you've done in the project.
- **Custom project IDs are lost.** When you created this project, you might have created a custom project ID that you want to use in the future. To preserve the URLs that use the project ID, such as an `appspot.com` URL, delete selected resources inside the project instead of deleting the whole project.

1. In the Cloud Console, go to the **Manage resources** page.

   GO TO THE MANAGE RESOURCES PAGE (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/IAM-ADMIN/PRC

2. In the project list, select the project you want to delete and click **Delete** .
3. In the dialog, type the project ID, and then click **Shut down** to delete the project.

What's next

- Configure fine-grained access control for Container Registry (https://cloud.google.com/container-registry/docs/access-control).
- Learn how to deploy a highly available SQL Server group on Compute Engine (https://cloud.google.com/solutions/deploy-multi-subnet-sql-server).
- Read about .NET on Google Cloud Platform (https://cloud.google.com/dotnet/docs/).
- Install Cloud Tools for Visual Studio (https://cloud.google.com/visual-studio/).
- Try out other Google Cloud Platform features for yourself. Have a look at our tutorials (https://cloud.google.com/docs/tutorials).

Last updated November 26, 2019.