Blue-Green Deployment in Azure Container Apps

Article06/28/20236 contributors Feedback

Select how you want to interact with Azure

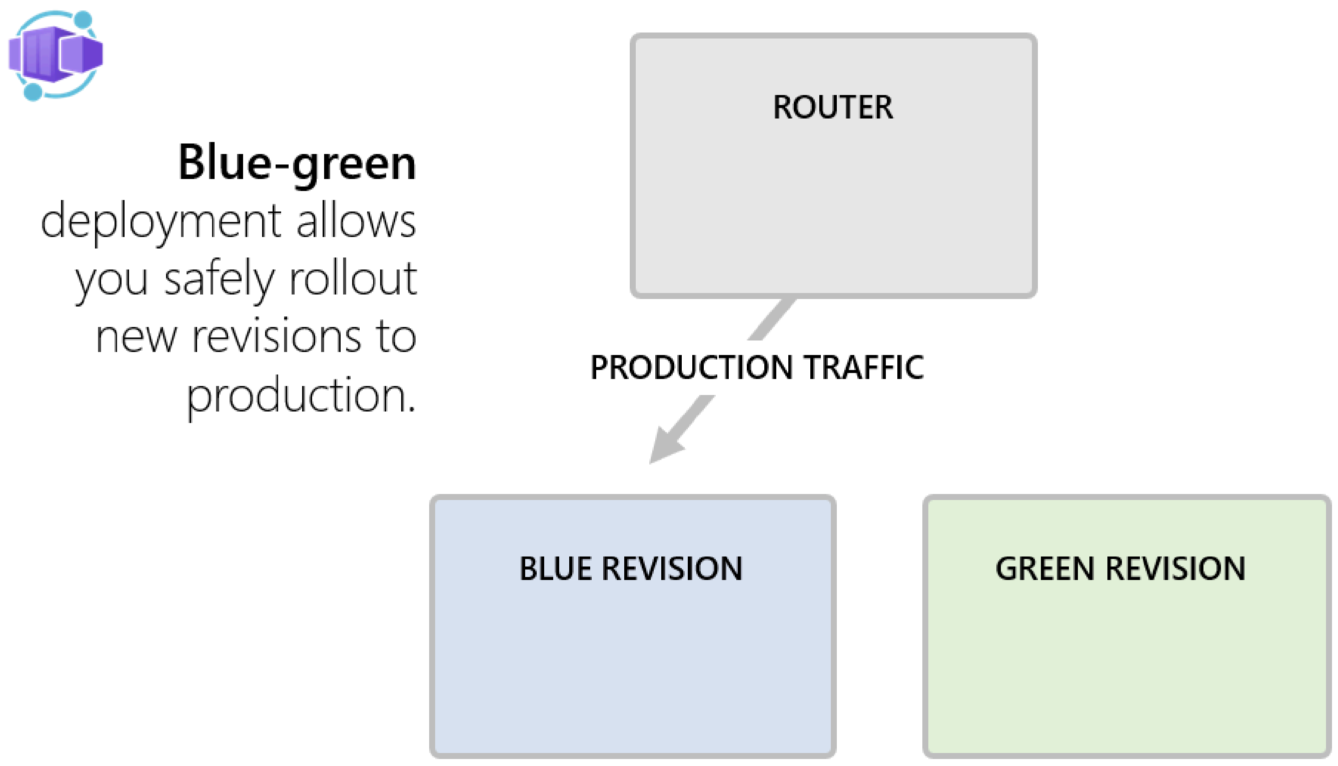
Azure CLI | Bicep

In this article

Blue-Green Deployment is a software release strategy that aims to minimize downtime and reduce the risk associated with deploying new versions of an application. In a blue-green deployment, two identical environments, referred to as "blue" and "green," are set up. One environment (blue) is running the current application version and one environment (green) is running the new application version.

Once green environment is tested, the live traffic is directed to it, and the blue environment is used to deploy a new application version during next deployment cycle.

You can enable blue-green deployment in Azure Container Apps by combining container apps revisions, traffic weights, and revision labels.


Blue-green deployment allows you safely rollout new revisions to production.
ROUTER
PRODUCTION TRAFFIC
BLUE REVISION
GREEN REVISION

You use revisions to create instances of the blue and green versions of the application.

Expand table

Revision	Description
Blue revision	The revision labeled as *blue* is the currently running and stable version of the application. This revision is the one that users interact with, and it's the target of production traffic.
Green revision	The revision labeled as *green* is a copy of the *blue* revision except it uses a newer version of the app code and possibly new set of environment variables. It doesn't receive any production traffic initially but is accessible via a labeled fully qualified domain name (FQDN).

After you test and verify the new revision, you can then point production traffic to the new revision. If you encounter issues, you can easily roll back to the previous version.

Expand table

Actions	Description
Testing and verification	The *green* revision is thoroughly tested and verified to ensure that the new version of the application functions as expected. This testing might involve various tasks, including functional tests, performance tests, and compatibility checks.
Traffic switch	Once the *green* revision passes all the necessary tests, a traffic switch is performed so that the *green* revision starts serving production load. This switch is done in a controlled manner, ensuring a smooth transition.
Rollback	If problems occur in the *green* revision, you can revert the traffic switch, routing traffic back to the stable *blue* revision. This rollback ensures minimal impact on users if there are issues in the new version. The *green* revision is still available for the next deployment.
Role change	The roles of the blue and green revisions change after a successful deployment to the *green* revision. During the next release cycle, the *green* revision represents the stable production environment while the new version of the application code is deployed and tested in the *blue* revision.

This article shows you how to implement blue-green deployment in a container app. To run the following examples, you need a container app environment where you can create a new app.

Note

Refer to **containerapps-blue-green repository** for a complete example of a GitHub workflow that implements blue-green deployment for Container Apps.

Create a container app with multiple active revisions enabled

The container app must have the `configuration.activeRevisionsMode` property set to `multiple` to enable traffic splitting. To get deterministic revision names, you can set the `template.revisionSuffix` configuration setting to a string value that uniquely identifies a release. For example you can use build numbers, or git commits short hashes.

For the following commands, a set of commit hashes was used.

Save the following code into a file named `main.bicep`.

Bicep

Copy

```bicep
targetScope = 'resourceGroup'
param location string = resourceGroup().location

@minLength(1)
```

```
@maxLength(64)
@description('Name of containerapp')
param appName string

@minLength(1)
@maxLength(64)
@description('Container environment name')
param containerAppsEnvironmentName string

@minLength(1)
@maxLength(64)
@description('CommitId for blue revision')
param blueCommitId string

@maxLength(64)
@description('CommitId for green revision')
param greenCommitId string = ''

@maxLength(64)
@description('CommitId for the latest deployed revision')
param latestCommitId string = ''

@allowed([
  'blue'
  'green'
])
@description('Name of the label that gets 100% of the traffic')
param productionLabel string = 'blue'

var currentCommitId = !empty(latestCommitId) ? latestCommitId : blueCommitId

resource containerAppsEnvironment 'Microsoft.App/managedEnvironments@2022-03-01' existing = {
  name: containerAppsEnvironmentName
}

resource blueGreenDeploymentApp 'Microsoft.App/containerApps@2022-11-01-preview' = {
  name: appName
  location: location
  tags: {
    blueCommitId: blueCommitId
    greenCommitId: greenCommitId
    latestCommitId: currentCommitId
    productionLabel: productionLabel
  }
  properties: {
    environmentId: containerAppsEnvironment.id
    configuration: {
      maxInactiveRevisions: 10 // Remove old inactive revisions
      activeRevisionsMode: 'multiple' // Multiple active revisions mode is required when using traffic weigh
      ingress: {
        external: true
        targetPort: 80
        traffic: !empty(blueCommitId) && !empty(greenCommitId) ? [
          {
            revisionName: '${appName}--${blueCommitId}'
            label: 'blue'
            weight: productionLabel == 'blue' ? 100 : 0
          }
          {
            revisionName: '${appName}--${greenCommitId}'
            label: 'green'
            weight: productionLabel == 'green' ? 100 : 0
          }
        ] : [
          {
            revisionName: '${appName}--${blueCommitId}'
```

```
              label: 'blue'
              weight: 100
            }
          ]
        }
      }
      template: {
        revisionSuffix: currentCommitId
        containers:[
          {
            image: 'mcr.microsoft.com/k8se/samples/test-app:${currentCommitId}'
            name: appName
            resources: {
              cpu: json('0.5')
              memory: '1.0Gi'
            }
            env: [
              {
                name: 'REVISION_COMMIT_ID'
                value: currentCommitId
              }
            ]
          }
        ]
      }
    }
  }

output fqdn string = blueGreenDeploymentApp.properties.configuration.ingress.fqdn
output latestRevisionName string = blueGreenDeploymentApp.properties.latestRevisionName
```

Deploy the app with the Bicep template using this command:

Azure CLI

Copy

```
export APP_NAME=<APP_NAME>
export APP_ENVIRONMENT_NAME=<APP_ENVIRONMENT_NAME>
export RESOURCE_GROUP=<RESOURCE_GROUP>

# A commitId that is assumed to belong to the app code currently in production
export BLUE_COMMIT_ID=fb699ef
# A commitId that is assumed to belong to the new version of the code to be deployed
export GREEN_COMMIT_ID=c6f1515

# create a new app with a blue revision
az deployment group create \
    --name createapp-$BLUE_COMMIT_ID \
    --resource-group $RESOURCE_GROUP \
    --template-file main.bicep \
    --parameters appName=$APP_NAME blueCommitId=$BLUE_COMMIT_ID containerAppsEnvironmentName=$APP_ENVIRONMEN
    --query properties.outputs.fqdn
```

Deploy a new revision and assign labels

The *blue* label currently refers to a revision that takes the production traffic arriving on the app's FQDN. The *green* label refers to a new version of an app that is about to be rolled out into production. A new commit hash identifies the new version of the app code. The following command deploys a new revision for that commit hash and marks it with *green* label.

Azure CLI

Copy

```
#deploy a new version of the app to green revision
az deployment group create \
    --name deploy-to-green-$GREEN_COMMIT_ID \
    --resource-group $RESOURCE_GROUP \
    --template-file main.bicep \
    --parameters appName=$APP_NAME blueCommitId=$BLUE_COMMIT_ID greenCommitId=$GREEN_COMMIT_ID latestCommitI
    --query properties.outputs.fqdn
```

The following example shows how the traffic section is configured. The revision with the *blue* `commitId` is taking 100% of production traffic while the newly deployed revision with *green* `commitId` doesn't take any production traffic.

JSON

Copy

```
{
  "traffic": [
    {
      "revisionName": "<APP_NAME>--fb699ef",
      "weight": 100,
      "label": "blue"
    },
    {
      "revisionName": "<APP_NAME>--c6f1515",
      "weight": 0,
      "label": "green"
    }
  ]
}
```

The newly deployed revision can be tested by using the label-specific FQDN:

Azure CLI

Copy

```
#get the containerapp environment default domain
export APP_DOMAIN=$(az containerapp env show -g $RESOURCE_GROUP -n $APP_ENVIRONMENT_NAME --query properties.

#Test the production FQDN
curl -s https://$APP_NAME.$APP_DOMAIN/api/env | jq | grep COMMIT

#Test the blue lable FQDN
curl -s https://$APP_NAME---blue.$APP_DOMAIN/api/env | jq | grep COMMIT

#Test the green lable FQDN
curl -s https://$APP_NAME---green.$APP_DOMAIN/api/env | jq | grep COMMIT
```

Send production traffic to the green revision

After confirming that the app code in the *green* revision works as expected, 100% of production traffic is sent to the revision. The *green* revision now becomes the production revision.

Azure CLI

Copy

```
# make green the prod revision
az deployment group create \
    --name make-green-prod-$GREEN_COMMIT_ID \
    --resource-group $RESOURCE_GROUP \
    --template-file main.bicep \
    --parameters appName=$APP_NAME blueCommitId=$BLUE_COMMIT_ID greenCommitId=$GREEN_COMMIT_ID latestCommit
    --query properties.outputs.fqdn
```

The following example shows how the `traffic` section is configured after this step. The *green* revision with the new application code takes all the user traffic while *blue* revision with the old application version doesn't accept user requests.

JSON Copy

```json
{
  "traffic": [
    {
      "revisionName": "<APP_NAME>--fb699ef",
      "weight": 0,
      "label": "blue"
    },
    {
      "revisionName": "<APP_NAME>--c6f1515",
      "weight": 100,
      "label": "green"
    }
  ]
}
```

Roll back the deployment if there were problems

If after running in production, the new revision is found to have bugs, you can roll back to the previous good state. After the rollback, 100% of the traffic is sent to the old version in the *blue* revision and that revision is designated as the production revision again.

Azure CLI Copy

```
# rollback traffic to blue revision
az deployment group create \
    --name rollback-to-blue-$GREEN_COMMIT_ID \
    --resource-group $RESOURCE_GROUP \
    --template-file main.bicep \
    --parameters appName=$APP_NAME blueCommitId=$BLUE_COMMIT_ID greenCommitId=$GREEN_COMMIT_ID latestCommit
    --query properties.outputs.fqdn
```

After the bugs are fixed, the new version of the application is deployed as a *green* revision again. The *green* version eventually becomes the production revision.

Next deployment cycle

Now the *green* label marks the revision currently running the stable production code.

During the next deployment cycle, the *blue* identifies the revision with the new application version being rolled out to production.

The following commands demonstrate how to prepare for the next deployment cycle.

Azure CLI Copy

```
# set the new commitId
export BLUE_COMMIT_ID=ad1436b
```

```
# deploy new version of the app to blue revision
az deployment group create \
    --name deploy-to-blue-$BLUE_COMMIT_ID \
    --resource-group $RESOURCE_GROUP \
    --template-file main.bicep \
    --parameters appName=$APP_NAME blueCommitId=$BLUE_COMMIT_ID greenCommitId=$GREEN_COMMIT_ID latestCommitI
    --query properties.outputs.fqdn
```

Next steps

Traffic Weights

Feedback