# Pipelines

## TLDR;

We use a single CICD azure-pipelines.yml for each microservice or shared library. Built microservices (images) will end up being hosted in an Azure Container Registry whilst C# shared libraries will be hosted as nugets on an Azure Feed. For a release, create a release branch in Azure DevOps with a named version e.g., ‘v0.1.2’ and manually trigger it.

# Next Steps - Suggestions

* trigger release pipeline on branch creation (not manually)
* introduce /bugfix path just like /feature
* switch to docker auth in pipelines – seems to be the direction of travel
* approval gates (integrates with Azure DevOps ‘Environments’)
* beyond ‘dev’ requirements are unknown – gather them!

## Approach

I have roughly followed these two links (updated November 2021) from the Microsoft architecture centre. Graphics are also from these links.

[Building a CI/CD pipeline for microservices on Kubernetes](https://docs.microsoft.com/en-us/azure/architecture/microservices/ci-cd-kubernetes)

[CI/CD for microservices architectures](https://docs.microsoft.com/en-us/azure/architecture/microservices/ci-cd-kubernetes)

I have used the new Kubernetes Manifest Tasks to deploy to k8s rather than Helm Tasks (based on Helm v2?). The approval gate is in Next Steps. I have used our decisions above:

* The code repository is a monorepo, with folders organised by microservice (Service A, Service B and a WebApp)
* The branching strategy is based on trunk-based development.
* The release strategy uses release branches to manage releases. Separate releases are created for each microservice e.g., Service A, Service B etc.
* The CI/CD process uses Azure Pipelines to build, test, and deploy the microservices to AKS.
* The container images for each microservice are stored in an Azure Container Registry.
* The new Kubernetes Manifest Task is used in the pipeline to deploy containers from the ACR to AKS for the sample Service A microservice.

## Build Pipeline

A developer is working on a microservice called the Service A. While developing a new feature, the developer checks code into a feature branch. This can be created either in Azure DevOps Boards directly on a work item, or by the Git command. By convention and by necessity for the demo to work, feature branches are named ***feature/servicea\**** e.g., feature/servicea/8150

Timeline

Description automatically generated

The build definition file includes a trigger that filters by the branch name and the source path. Using this approach, each microservice can have its own build pipeline. Only code that is checked into the **/Services/ServiceA/** folder triggers a build of Service A. Pushing commits to a branch that matches the filter triggers a CI build. At this point in the workflow, the CI build runs some minimal code verification:

1. Build the code.
2. Run unit tests. (<todo>)

The goal is to keep build times short, so the developer can get quick feedback. Once the feature is ready to merge into main, the developer opens a PR from this feature branch into main.

At this point we could trigger another CI build that performs some additional checks:

1. Build the code.
2. Run unit tests.
3. Build the runtime container image.

I have not put these in but since they are a superset of what has been done it would be easy to incorporate them.

## Release Pipeline

At some point, we are ready to deploy a new version of Service A. The release manager creates a branch from main with this naming pattern:

release/<microservice name>/<semver>. For example, release/servicea/v1.14.0.

See ‘semantic versioning if you’re not familiar’

Chart

Description automatically generated

Creation of this branch triggers a full CI build that runs all of the previous steps plus:

1. Push the container image to Azure Container Registry. The image is tagged with the version number – right now it’s the build number but it should come from the branch name. <todo>
2. Approval stage (<todo>)

Assuming this build succeeds, it triggers a deployment (CD) process using an Azure Pipelines release pipeline. This pipeline has the following steps:

1. Run the k8s manifest tasks to deploy the service.

Once we get our environments sorted out, we can introduce approver steps and the retagging of the Docker image for the production namespace in Azure Container Registry. For example, if the current tag is canterbury.azurecr.io/servicea:v1.14.1, the production tag is canterbury.azurecr.io/prod/servicea:v1.14.1.

Even in a monorepo, these tasks can be scoped to individual microservices, so that we can deploy microservices with high velocity. The process has some manual steps: Approving PRs, creating release branches, and approving deployments into the production.

We use a manifests directory containing deployment.yml and ~~service.yml~~.

Each microservice requires other yaml files apart from the deployment file itself. The intention is to use Helm for that once we have something simple set up.

## Step-By-Step (for VS)

1. In Azure Devops Boards, add a dummy Issue and a dummy task. From the Issue, create a feature branch for this work following this convention feature/featurenumber e.g. feature/123
2. In VS, pull the latest branch in via a Fetch from the Team Explorer. You should now be able to Checkout the new feature branch and work on the changes
3. You can make a dummy change to the code – say in what one of the APSs return. Commit your change(s)
4. When you do a Push back to Azure DevOps, a trigger will be fired on an update to this branch (feature/\*) and this path (/Services/ServiceA/). This triggers a build and the <todo> running of unit tests either from AzureDevOps or from inside the docker container dotnet test…
5. When ready, create a pull request in VS <your branch> into main
6. In Azure DevOps go to Pull Requests and approve your own request (normally there should be a policy to stop you approving your own request). The new microservice code is now in main.
7. When the Release manager is ready to make a release, he/she creates a new release branch from main using the following convention release/servicea/v1.14.0
8. Creating this new branch should trigger a build but Azure DevOps removed that 2 years ago <todo> investigate it returning as Github has this exact feature
9. This triggers a new build, test and new image creation and push to an Azure Container Repository (ACR). The trigger is on a yaml override on a branch filter release/servicea/\*
10. The pipeline continues with a deployment to a Kubernetes cluster. For PROD, we should have a manual approval step.

# Notes

* Yaml – I prefer to work in VS Code because of the extensions support – Pipelines, Bicep, Docker etc.
* Docker context is relative to the path of the Dockerfile while the Visual Studio context is relative to the .sln path. Lots of fun.
* Helm – good for packaging up all the files to do with a microservice. We will need helm to package up the solution for partners and end-customers – probably on Docker hub
* Helm chart support still in preview for Azure Container Registry
* Azure Pipelines k8s manifest task is simpler than helm and works well with Azure Pipelines ‘Environments’ (great traceability)
* Pace of change of Azure DevOps is fast!