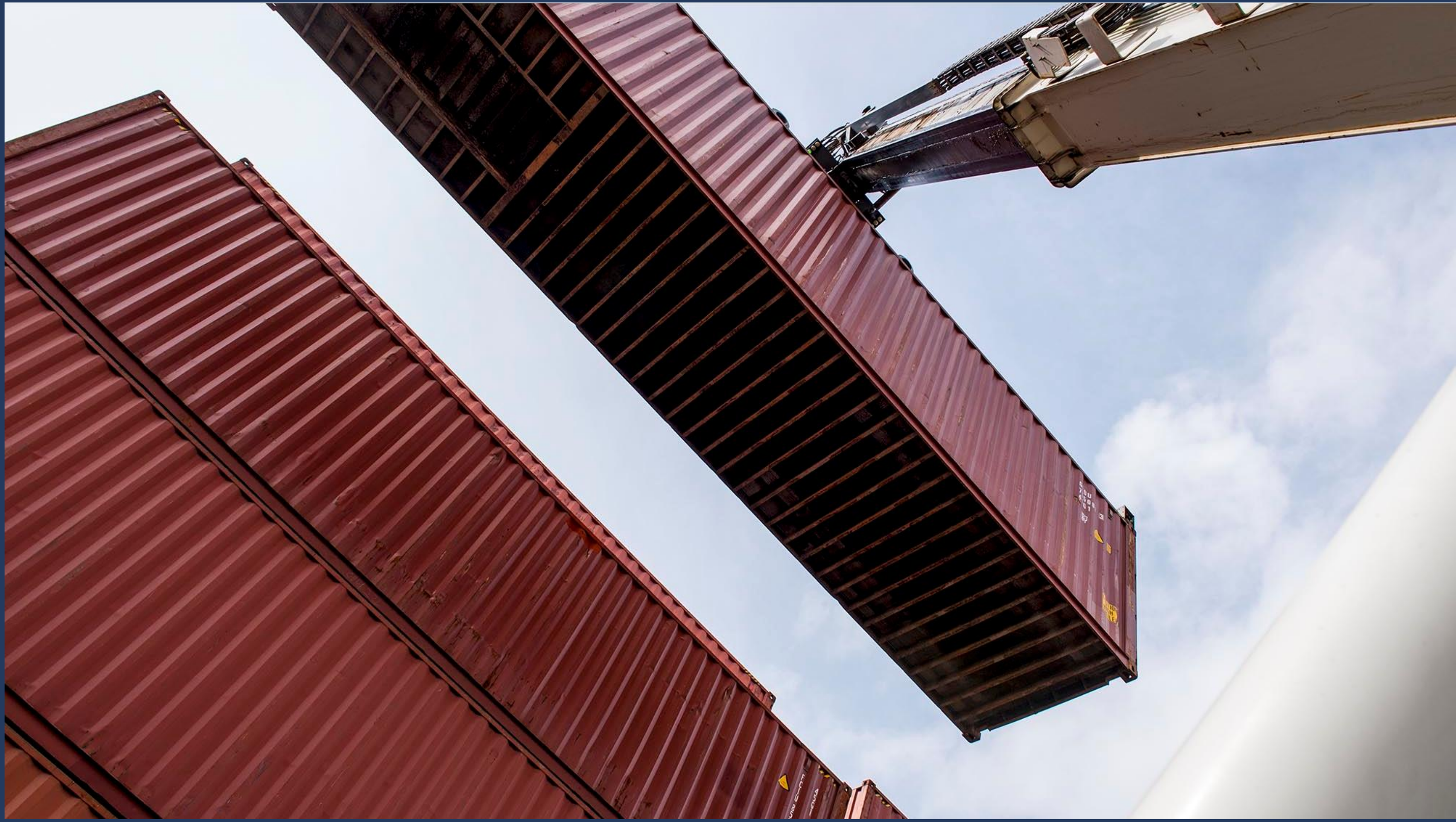
Microsoft

Kubernetes Technical Briefing

Module 5:
Application Deployment



Agenda

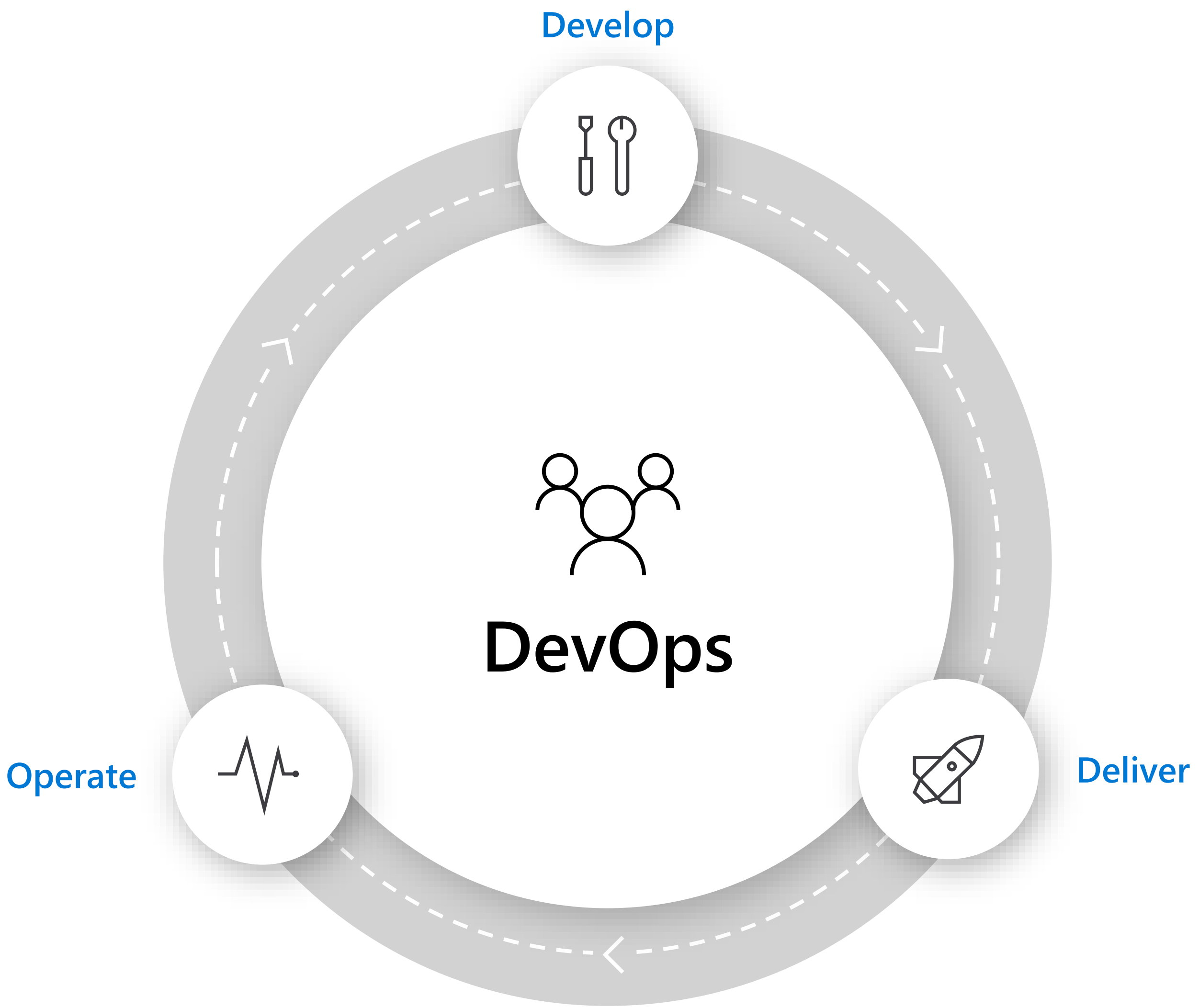
- DevOps and GitOps Concepts
- Basic Microservices Deployments
- Complex Microservices Deployments with Helm and Azure Key Vault

1

DevOps and GitOps Concepts

What is DevOps? People. Process. Products.

- DevOps is the union of **people**, **process**, and **products** to enable continuous delivery of value to your end users.
- This training modules covers only basic functionality of tools like ***Azure DevOps*** and ***GitHub***.
- Please refer to our other Microsoft DevOps workshops for more details.

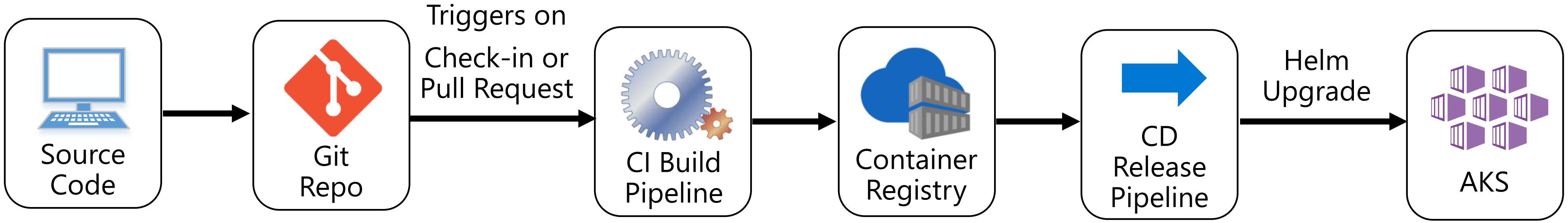


What is GitOps?

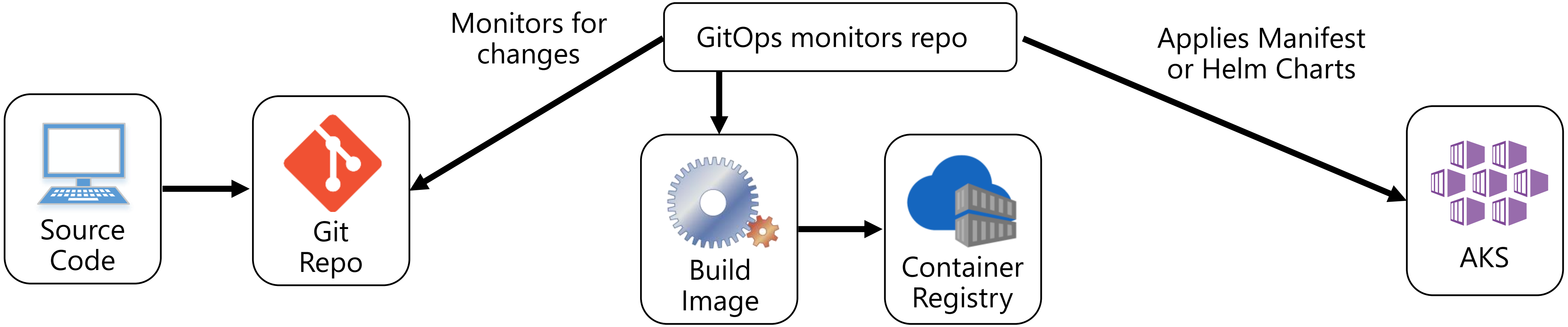
- GitOps is an operational framework that takes DevOps best practices used for application development such as version control, collaboration, compliance, and CI/CD, and applies them to infrastructure automation.
- GitOps uses a ***Git repository*** as the single source of truth for infrastructure definitions.
- GitOps automates infrastructure updates using a Git workflow with continuous integration (CI) and continuous delivery (CD).

DevOps vs GitOps

- **DevOps** uses pipelines that ***Push*** changes to a destination cluster.



- **GitOps** monitors a Git repo and ***Pulls*** changes from the Git repo to synchronize a destination cluster.

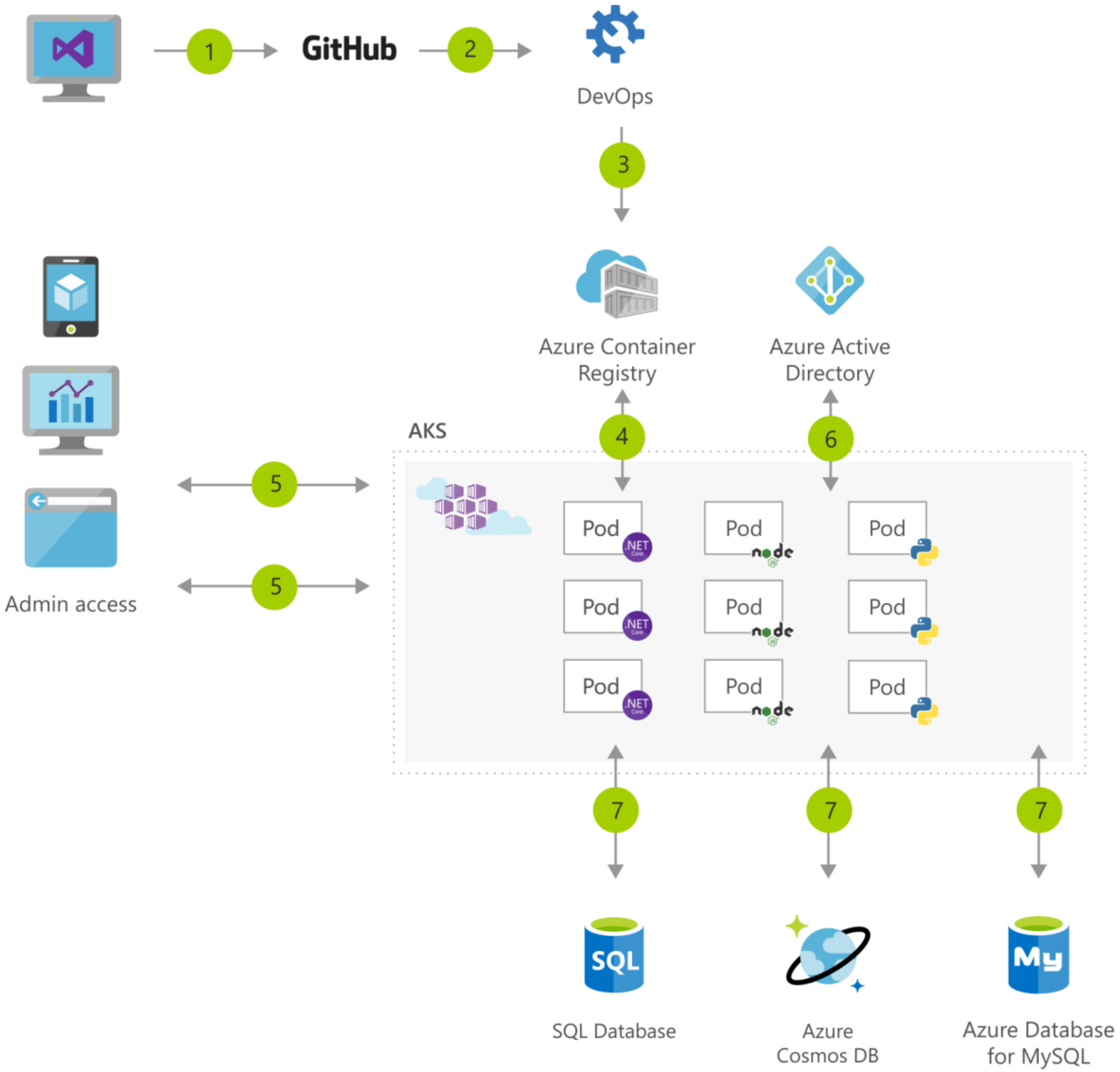


Basic Microservices Deployments

2

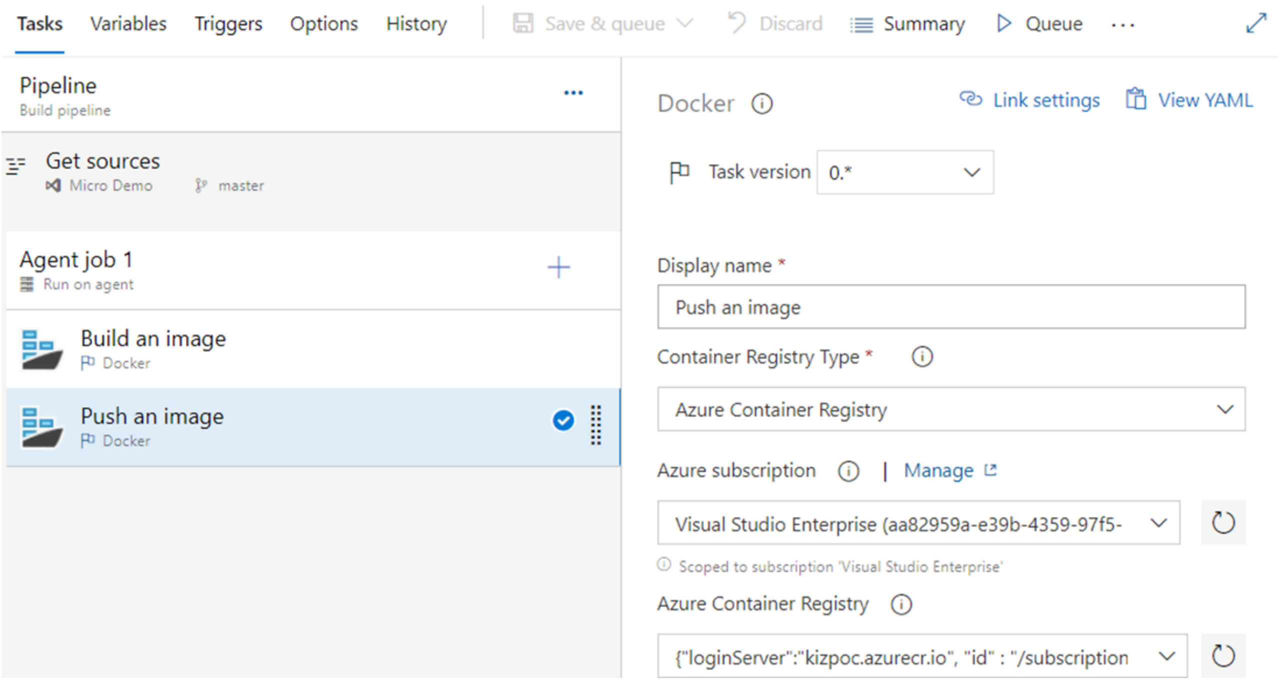
Microservices Architecture in Azure

1. Developer uses an IDE, such as Visual Studio, to commit changes to GitHub or Azure DevOps.
2. GitHub triggers a new build on Azure DevOps.
3. Azure DevOps packages the microservices as containers and pushes them to the Azure Container Registry.
4. Containers are deployed to the AKS cluster.
5. Users access services via apps and a website.
6. Azure Active Directory is used to secure access to the resources.
7. Microservices can use databases to store and retrieve information.



Azure DevOps Continuous Integration (CI)

- DevOps practices, such as Continuous Integration and Continuous Delivery, are used to drive microservice deployments.
- Use Azure DevOps ***Build Pipelines*** to automatically build and push images to an ACR every time when you check in your code.



Task version 0.*

Display name *
Push an image

Container Registry Type *
Azure Container Registry

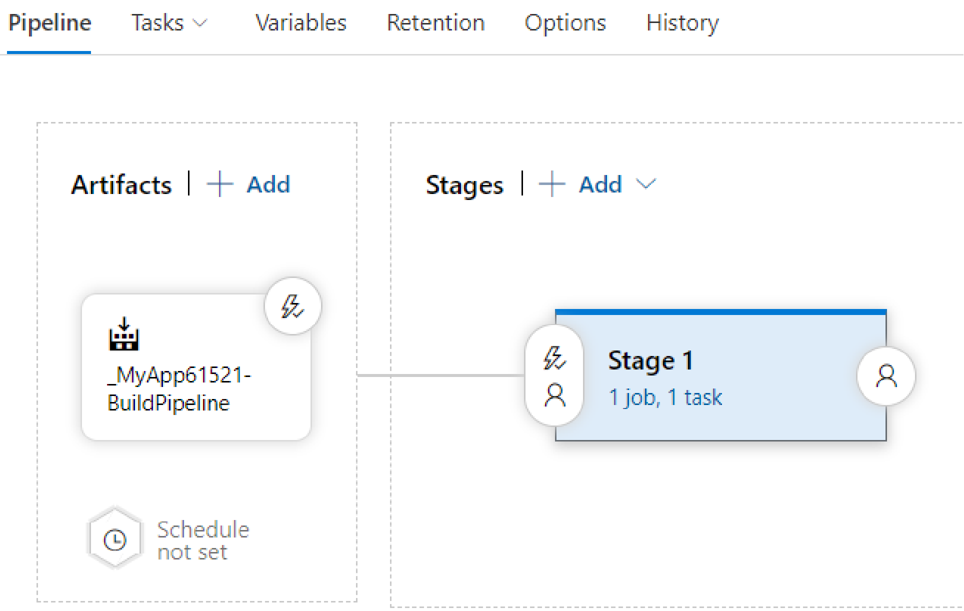
Azure subscription
Visual Studio Enterprise (aa82959a-e39b-4359-97f5-

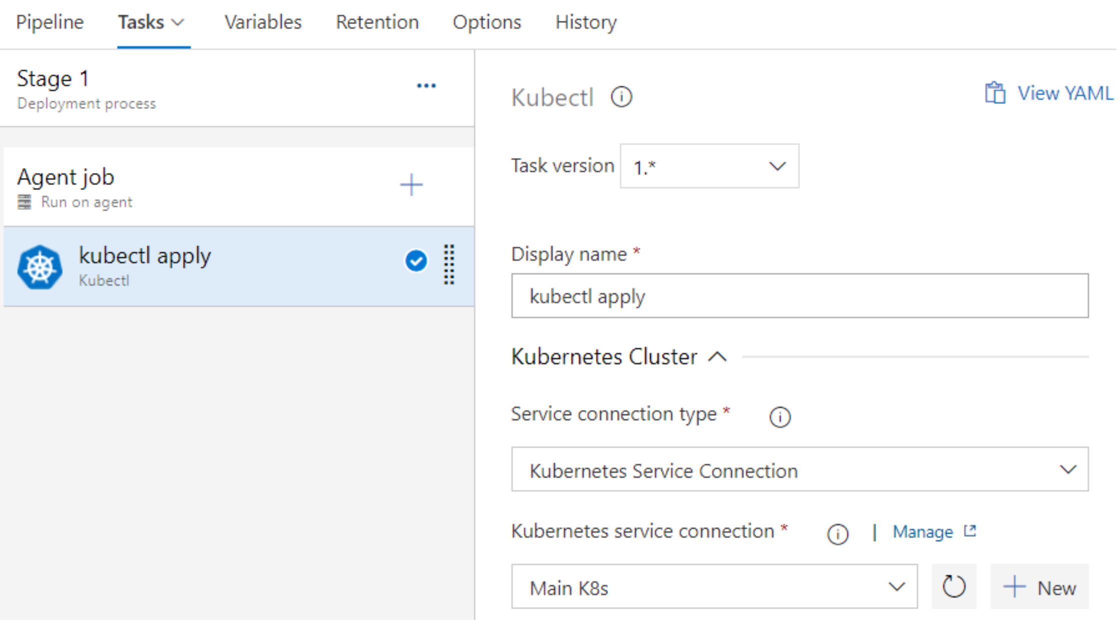
Scoped to subscription 'Visual Studio Enterprise'

Azure Container Registry
{"loginServer":"kizpoc.azurecr.io", "id" : "/subscription

Azure DevOps Continuous Delivery (CD)

Use Azure DevOps ***Release Pipelines*** to apply updated YAML files to your AKS cluster.





Task version 1.*

Display name * kubectl apply

Service connection type * Kubernetes Service Connection

Kubernetes service connection * Main K8s

Basic Azure DevOps CI/CD Process

The basic Azure DevOps CI/CD process consists of the following parts:

- **Code Repository** – Contains the microservice source code.
- **Build Pipeline** – Triggers when source code is checked in or a pull request is made.
  - A Build Number is used to uniquely identify the image about to be built. Can be built-in or generated.
  - A Docker *build* task builds an image based on the Dockerfile included with the source code. It uses the Build Number (or variation of it) as the image's tag.
  - A Docker *push* task pushes the image to a container registry
  - A RegEx task updates the deployment manifest (*yaml* file), by replacing the "*:latest*" string with the tag used during the build. This is how Kubernetes knows which image to deploy.
  - A Publish task saves the updated deployment manifest in a zip file, known as an ***artifact***.
- **Release Pipeline** – Triggers when a Build completes
  - A *kubectl* task extracts the deployment manifest from the ***artifact*** and applies it to the cluster.

Basic Azure DevOps Demo

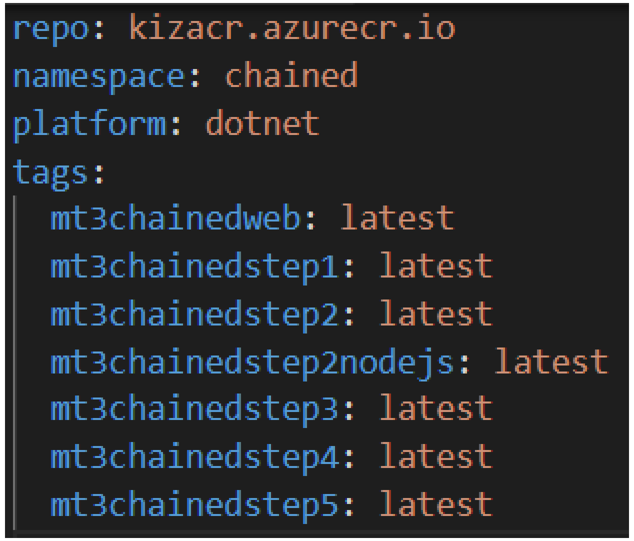
Using Azure CI/CD to build and push images to ACR and update AKS

3

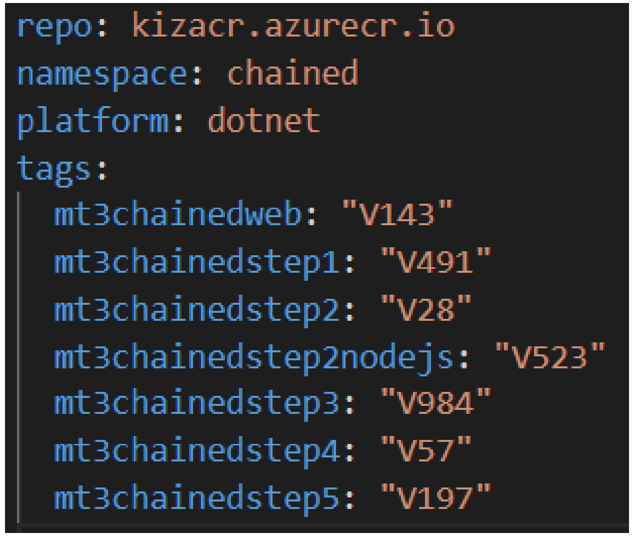
Complex Microservices Deployments with Helm and Azure Key Vault

Complex Deployments with Azure DevOps

- Complex applications, that utilize multiple microservices, need a more detailed deployment process.
- Rather than updating individual deployments using "**kubectl apply**", the CD process uses **Helm** to upgrade the application as a complete system.
- If the "latest" version of each microservice is deployed, then Helm can simply do an upgrade without any changes.
- However, it's best practice not to use the "latest" tag of any container (public or custom) in a production application.
- To correctly upgrade a chart, Helm needs to know the specific version of each microservice to deploy.
- Staging and production pipelines usually need to be able to deploy different versions of each microservice.



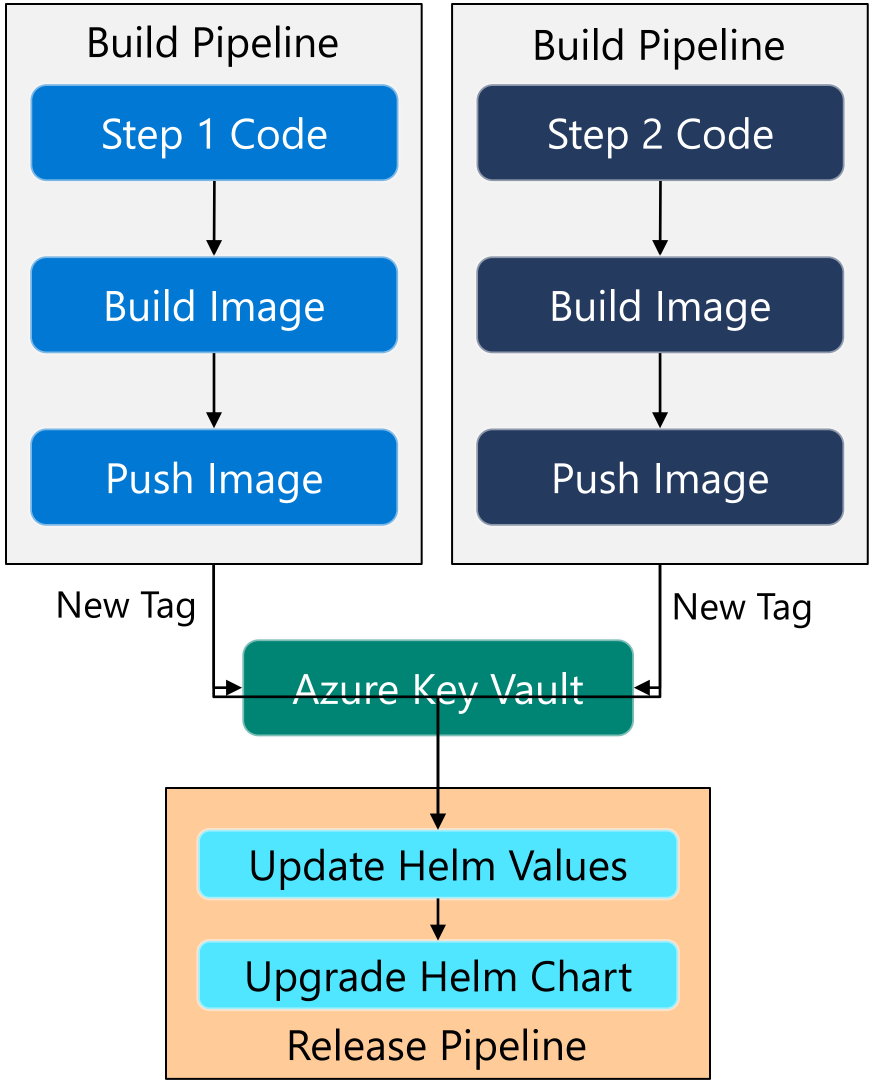
```
repo: kizacr.azurecr.io
namespace: chained
platform: dotnet
tags:
  mt3chainedweb: latest
  mt3chainedstep1: latest
  mt3chainedstep2: latest
  mt3chainedstep2nodejs: latest
  mt3chainedstep3: latest
  mt3chainedstep4: latest
  mt3chainedstep5: latest
```



```
repo: kizacr.azurecr.io
namespace: chained
platform: dotnet
tags:
  mt3chainedweb: "V143"
  mt3chainedstep1: "V491"
  mt3chainedstep2: "V28"
  mt3chainedstep2nodejs: "V523"
  mt3chainedstep3: "V984"
  mt3chainedstep4: "V57"
  mt3chainedstep5: "V197"
```

Parallel Deployment Process

- A Build pipeline generates a tag specific to that build.
- A single Release pipeline can trigger when ANY microservice Build completes.
  - Since the same Helm chart is upgraded every time, there's no reason to have a separate Release pipeline per service.
- However, a release pipeline that uses Helm can't depend on tags being passed in as artifacts
  - It doesn't know which build triggered it.
  - It needs the tags of other services that were built earlier.
- The pipelines need a central location they can store and retrieve tags from, that is external to the CI/CD process.
- A good option for such a centralized storage is Azure Key Vault.

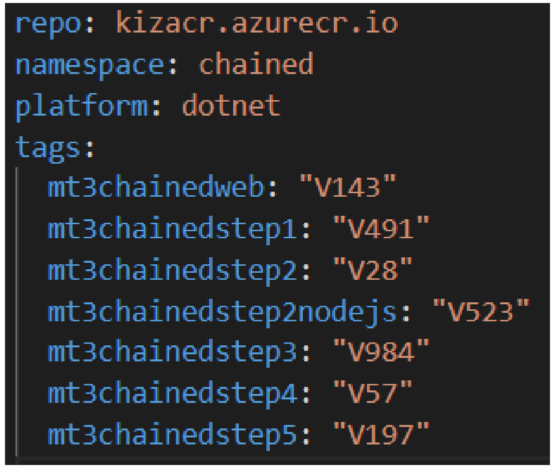


Microservice Azure DevOps CI/CD Process

The microservice Azure DevOps CI/CD process consists of the following parts:

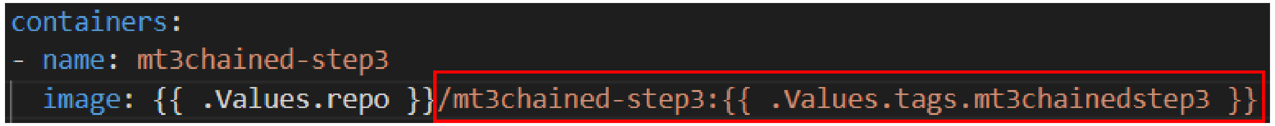
- **Code Repository** – Contains the microservice source code.
- **Build Pipeline** – Triggers when source code is checked in or a pull request is made.
  - A Build Number is used to uniquely identify the image about to be built. Can be built-in or generated.
  - A Docker *build* task builds an image based on the Dockerfile included with the source code. It uses the Build Number (or variation of it) as the image's tag.
  - A Docker *push* task pushes the image to a container registry.
  - An *Azure Key Vault* task stores the service name and tag as a key/value pair.
    - Example: ***mt3chained-step3=V124***
- **Release Pipeline** – Triggers when ANY associated Build completes
  - An *Azure Key Vault* task retrieves the tags of ALL related microservices.
  - A RegEx task updates all the tags in a copy of the **values.yaml** file.
  - A Helm task uses the updated values file to upgrade the chart:

```
helm upgrade myrelease mychart --install -f updatedvalues.yaml
```

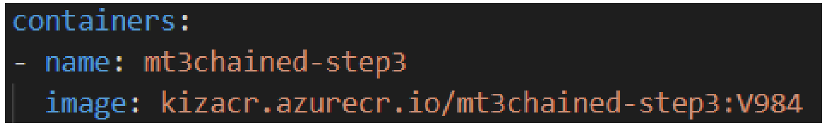


```
repo: kizacr.azurecr.io
namespace: chained
platform: dotnet
tags:
  mt3chainedweb: "V143"
  mt3chainedstep1: "V491"
  mt3chainedstep2: "V28"
  mt3chainedstep2nodejs: "V523"
  mt3chainedstep3: "V984"
  mt3chainedstep4: "V57"
  mt3chainedstep5: "V197"
```

updatedvalues.yaml

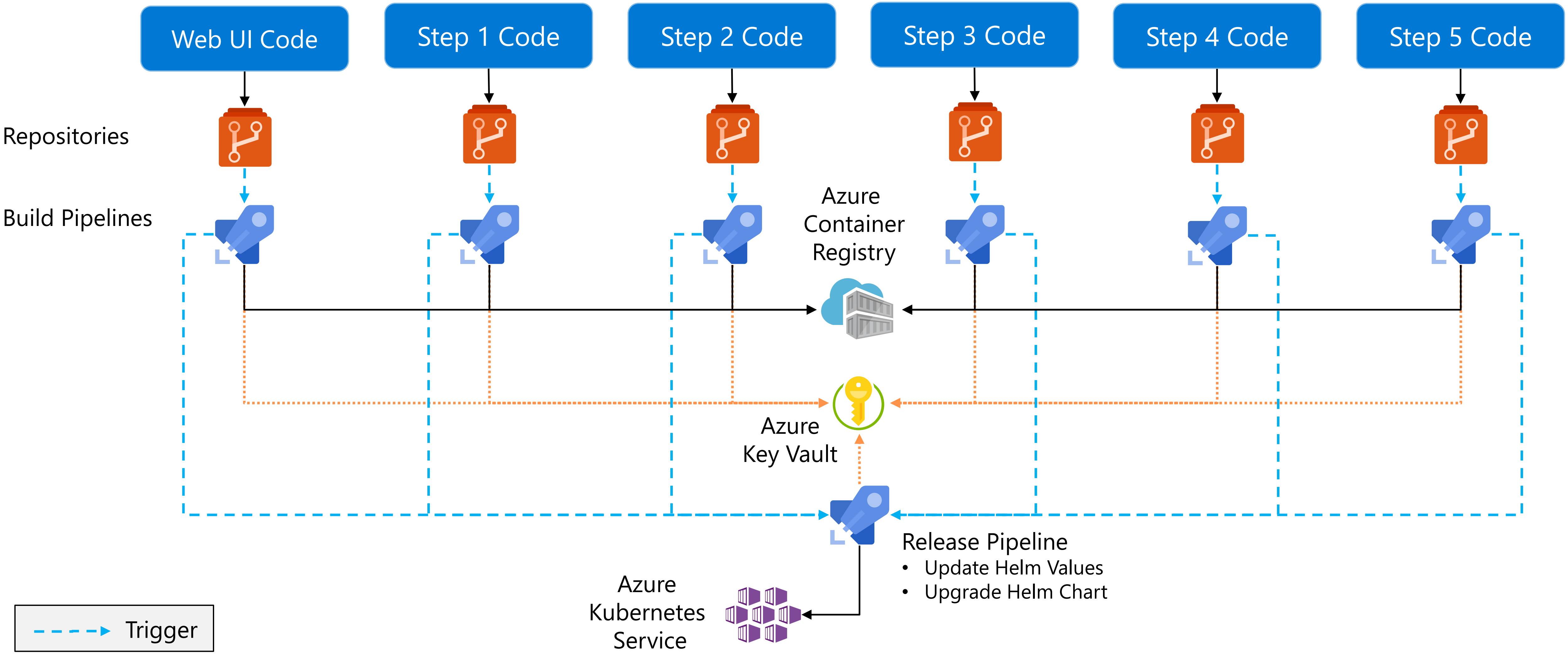


```
containers:
- name: mt3chained-step3
  image: {{ .Values.repo }}/mt3chained-step3:{{ .Values.tags.mt3chainedstep3 }}
```



```
containers:
- name: mt3chained-step3
  image: kizacr.azurecr.io/mt3chained-step3:V984
```

Overview of Microservices CI/CD Process



Lab – Module 5

Application Deployment to Kubernetes

Microsoft

Thank you