AZ-400.00 Learning Path 03: Implement CI with Azure Pipelines and GitHub Actions

Agenda

- Module 01: Explore Azure Pipelines.
- Module 02: Manage Azure Pipeline agents and pools.
- Module 03: Describe pipelines and concurrency.
- Module 04: Explore Continuous integration.
- Module 05: Implement a pipeline strategy.
- Module 06: Integrate with Azure Pipelines.
- Module 07: Introduction to GitHub Actions.
- Module 08: Learn continuous integration with GitHub Actions.
- Module 09: Design a container build strategy.
- Labs & Learning Path review and takeaways.

Learning Path overview

Learning objectives

After completing this learning path, students will be able to:

1. Explain the role of Azure Pipelines and its components
2. Implement a container strategy including how containers are different from virtual machines
3. Configure agents for use in Azure Pipelines
4. Explain why continuous integration matters

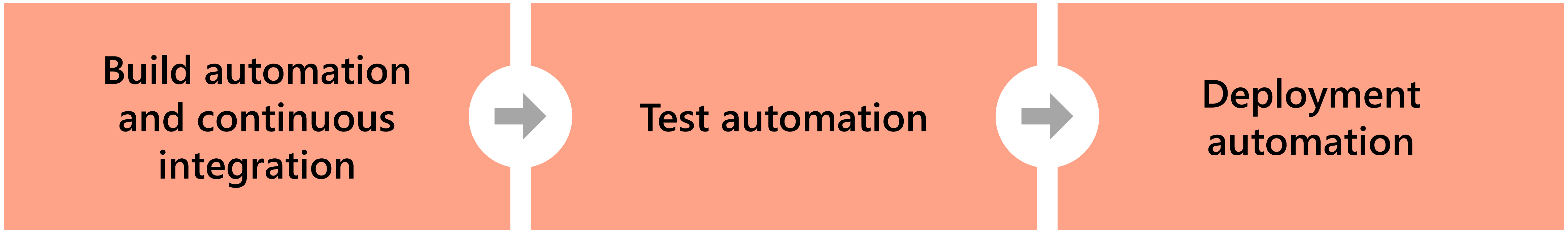
Learning objectives (continued)

After completing this Learning Path, students will be able to:

5. Implement continuous integration using Azure DevOps
6. Create and work with GitHub Actions and workflows
7. Implement Continuous Integration with GitHub Actions

Module 01: Explore Azure Pipelines

Explore the concept of pipelines in DevOps



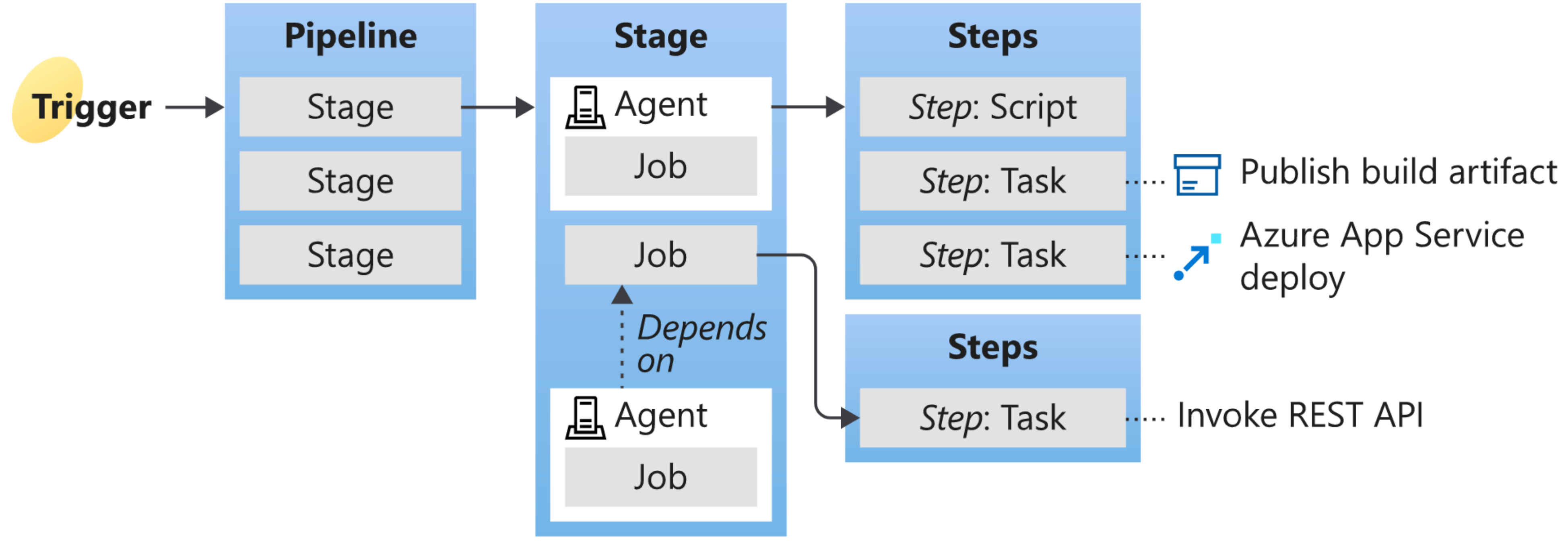
- A pipeline enables a constant flow of changes into production via an automated software production line
- Pipelines create a repeatable, reliable and incrementally improving process for taking software from concept to customer
- Pipelines require infrastructure, this infrastructure will have a direct impact on the effectiveness of the pipeline

Describe Azure Pipelines

1. Azure Pipelines is a cloud service that you can use to automatically build and test your code project and make it available to other users.

2. **Works great with Continuous Integration and Continuous Delivery:**
   - Work with any language or platform – Python, Java, PHP, Ruby, C#, and Go
   - Deploy to different types of targets at the same time
   - Integrate with Azure deployments – Container registries, virtual machines, Azure services, or any on-premises or cloud target (Microsoft Azure, Google Cloud, or AWS)
   - Build on Windows, Linux, or macOS machines
   - Integrate with GitHub
   - Work with open-source projects

Understand Azure Pipelines key terms



Continuous Delivery (CD)

Continuous Integration (CI)

Deployment target

Module 02: Manage Azure Pipeline agents and pools

Choose between Microsoft-hosted versus self-hosted agents

- **You generally need at least one agent to build or deploy your project.**
- **An agent is installable software that runs one build or deployment job at a time.**

Two types of agents:

1. **Microsoft-hosted agents** – Maintenance and upgrades are automatically done. Each time a pipeline is run, a fresh virtual machine (instance) is provided. There are time limits on jobs running on these agents.

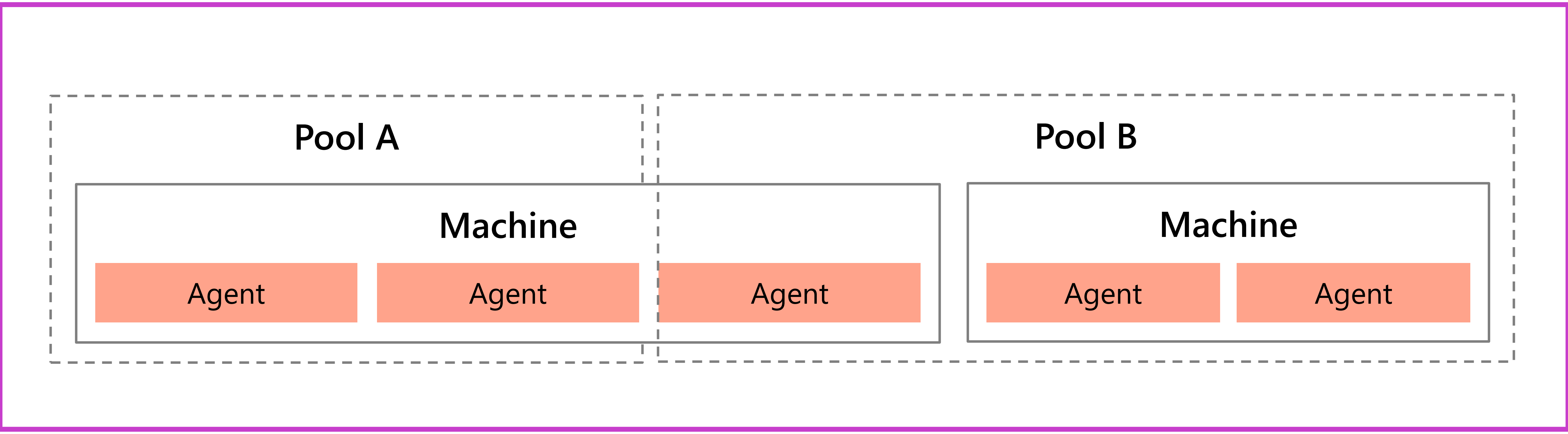
2. **Self-hosted agents** – You take care of maintenance and upgrades. Give you more control to install dependent software needed. Can be installed on Linux, macOS, Windows machines, or in a Linux Docker container. There are no time limits on jobs running on these agents.

Explore job types

Four types of jobs:

1. **Agent pool jobs** – Jobs that run on an agent in an agent pool
2. **Container jobs** – Jobs that run in a container on an agent in an agent pool
3. **Deployment group jobs** – Jobs that run on systems in a deployment group
4. **Agentless jobs** – Jobs that run on the Azure DevOps server (also called Server Jobs)

Introduction to agent pools



You can organize agents into agent pools.

An agent pool defines the sharing boundary.

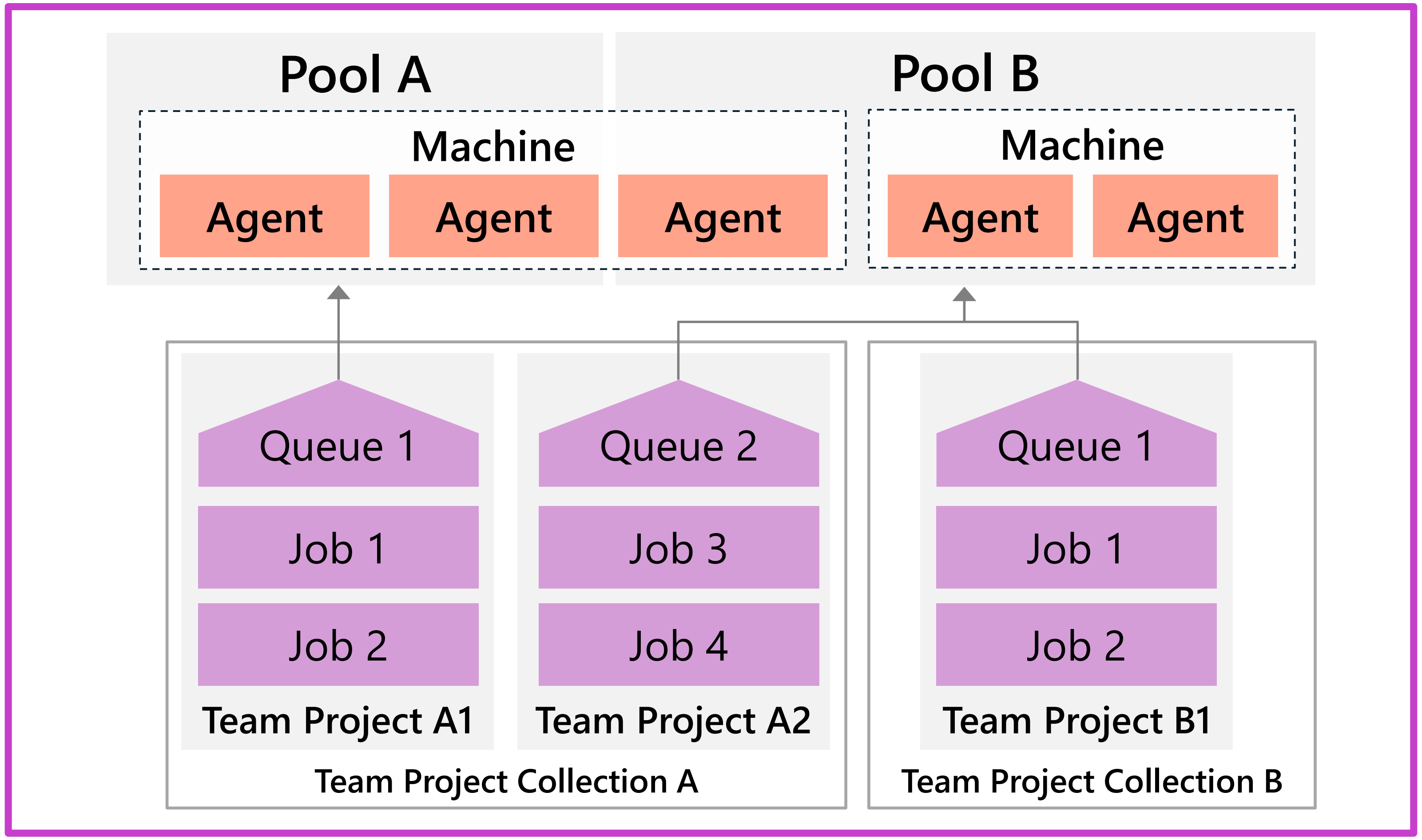
In Azure Pipelines, agent pools are scoped to the Azure DevOps organization; so, you can share an agent pool across projects.

Explore predefined agent pool

Image	Purpose	YAML VM Image Label
Windows Server 2022 with Visual Studio 2022	Windows-2022	Windows-latest OR windows-2022
Windows Server 2019 with Visual Studio 2019	Windows-2019	Windows-2019
Ubuntu 22.04	Ubuntu-22.04	Ubuntu-latest OR ubuntu-22.04
Ubuntu 20.04	Ubuntu-20.04	Ubuntu-20.04
macOS 12 Monterey	MacOS-12	Macos-latest OR macOS-12
macOS 11 Big Sur	MacOS-11	Macos-11

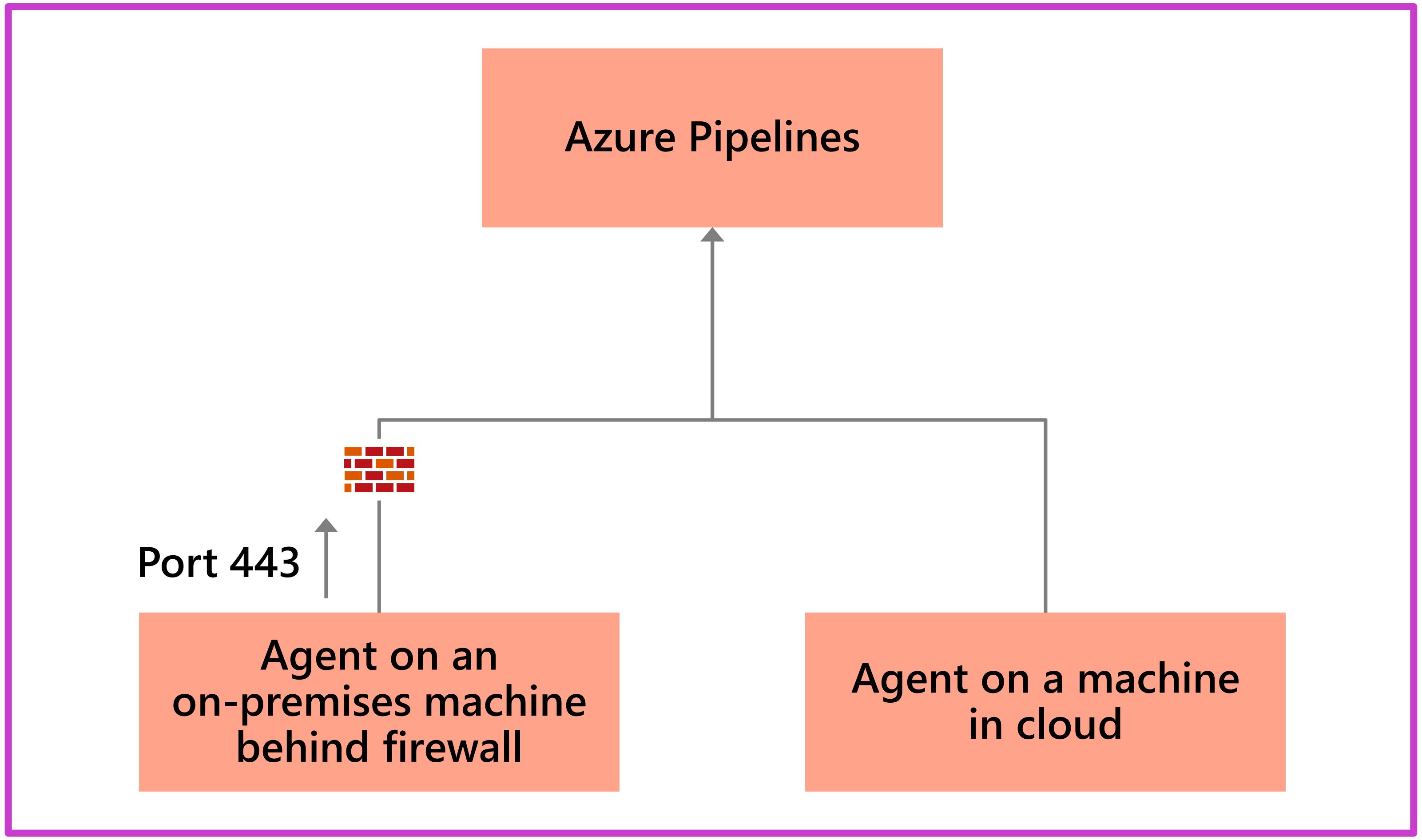
Understand typical situations for agent pools

- You're a member of a project and you want to use a set of machines owned by your team for running build and deployment jobs.
- You're a member of the infrastructure team and would like to set up a pool of agents for use in all projects.
- You want to share a set of agent machines with multiple projects but not all of them.



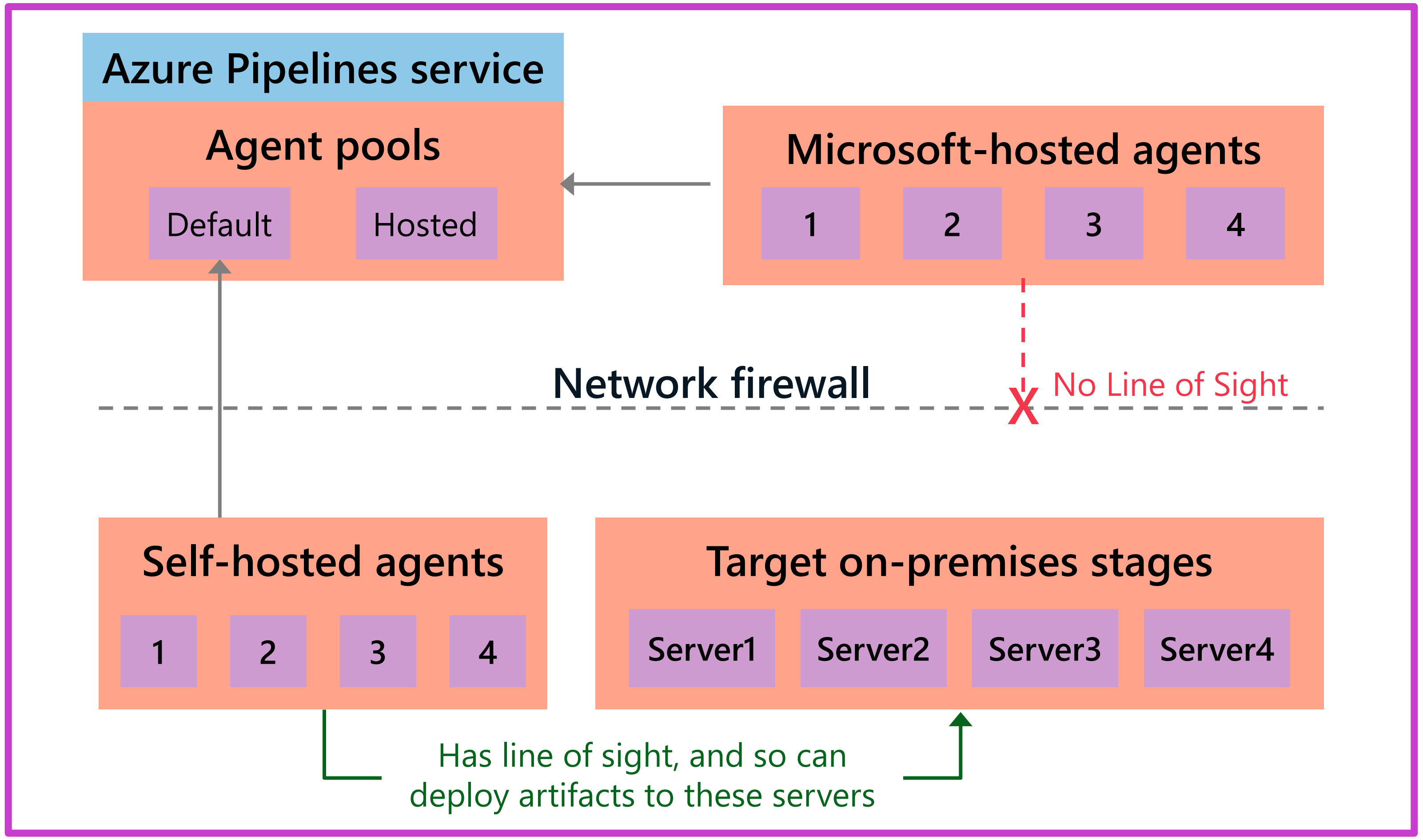
Communicate with Azure Pipelines

- The agent determines which job it needs to run, report the logs, and job status.
- Communication is always initiated by the agent.
- All the messages from the agent to Azure Pipelines are over HTTPS.


Azure Pipelines
Port 443
Agent on an on-premises machine behind firewall
Agent on a machine in cloud

Communicate to deploy to target servers

- Agent must have “line of sight” connectivity to servers.
- Microsoft-hosted agent pools, by default, have connectivity to Azure websites and servers running in Azure.
- You may need to manually configure connectivity.



Examine other considerations

1. Authentication
2. Personal Access Tokens
3. Interactive vs Service processes
4. Agent version and upgrades

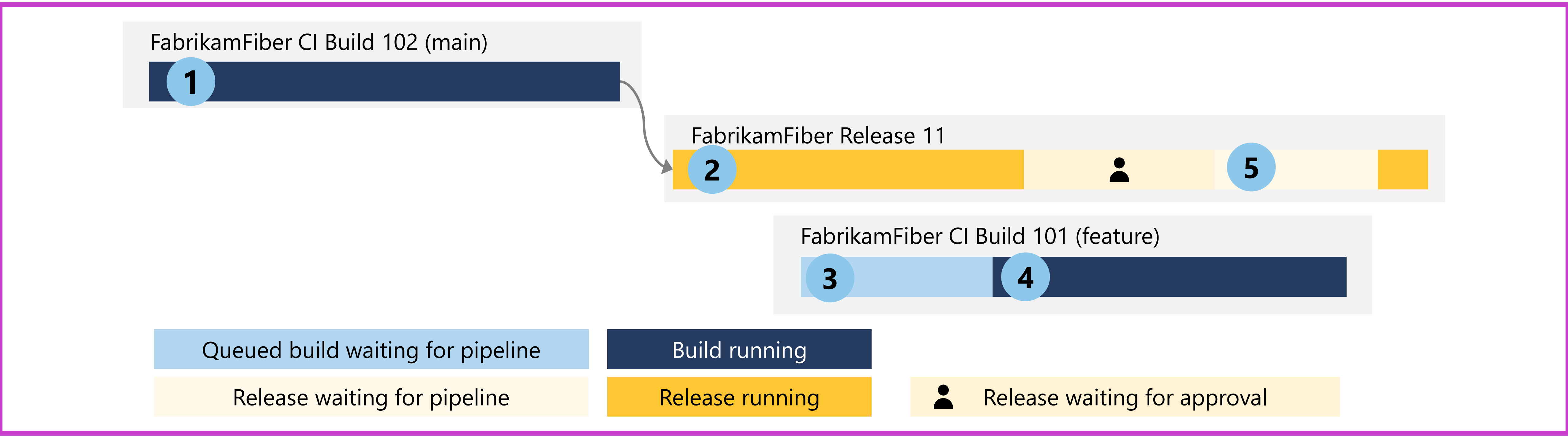
Describe security of agent pools

Roles are defined on each agent pool, and membership in these roles governs what operations you can perform on an agent pool.

Role	Purpose
Reader	Members of this role can view the organization agent pool as well as agents. You typically use this to add operators that are responsible for monitoring the agents and their health.
Service Account	Members of this role can use the organization agent pool to create a project agent pool in a project. If you follow the guidelines above for creating new project agent pools, you typically do not have to add any members here.
Administrator	In addition to all the above permissions, members of this role can register or unregister agents from the organization agent pool. They can also refer to the organization agent pool when creating a project agent pool. Finally, they can also manage membership for all roles of the organization agent pool. The user that created the organization agent pool is automatically added to the Administrator role.

Module 03: Describe pipelines and concurrency

Understand parallel jobs


FabrikamFiber CI Build 102 (main)
1
FabrikamFiber Release 11
2
5
FabrikamFiber CI Build 101 (feature)
3
4
Queued build waiting for pipeline
Build running
Release waiting for pipeline
Release running
Release waiting for approval

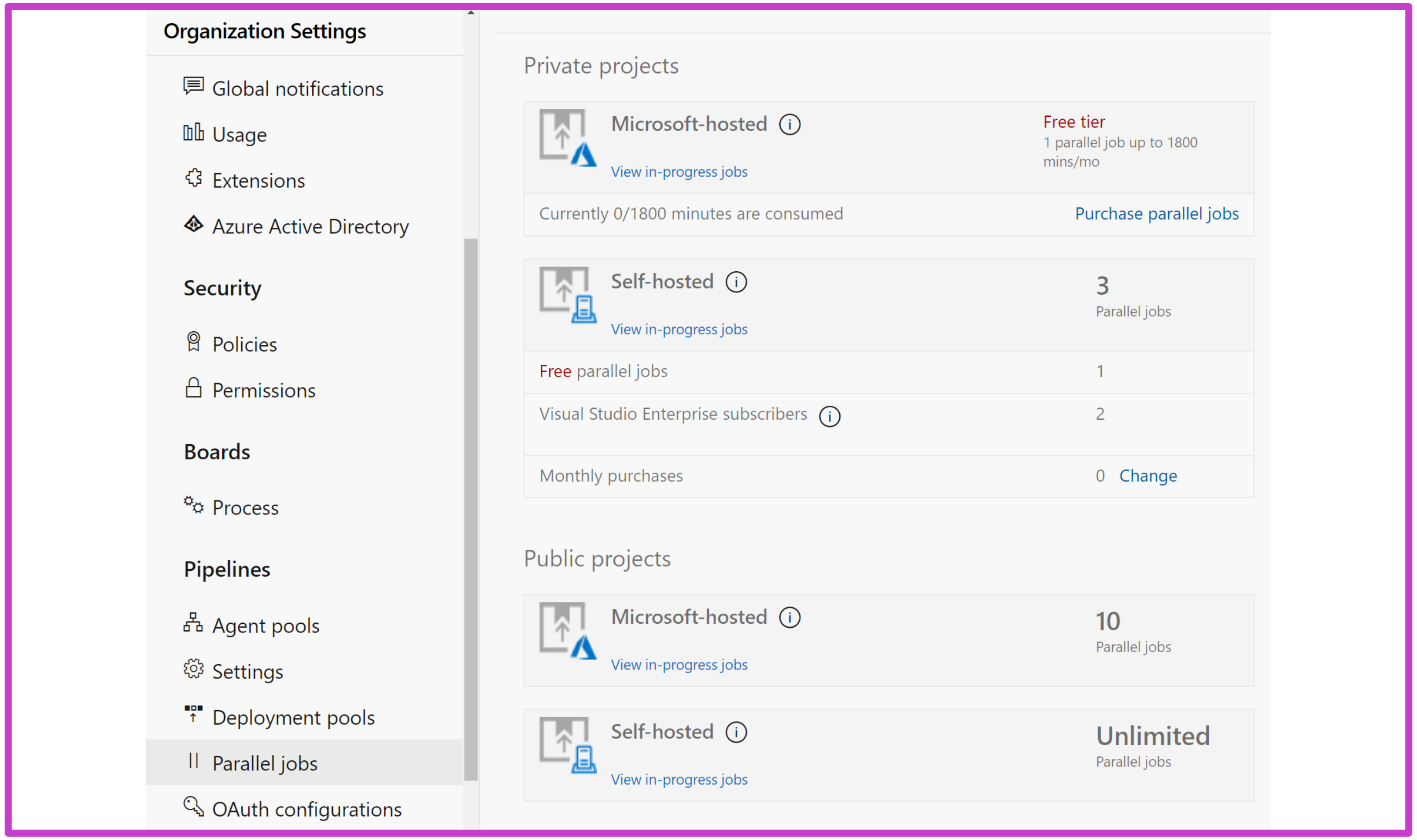
This is a simple example of a Microsoft-hosted parallel job.

Users can collectively run only one build or release job at a time; additional jobs are queued.

A release consumes a parallel job only when it's being actively deployed to a stage.

Estimate parallel jobs

- Determine how many parallel jobs you need.
- Simple estimates vs Detailed estimates
- You can display all the builds and releases.
- Parallel jobs are purchased at the organization level and are shared by all projects.



Private projects

Microsoft-hosted — View in-progress jobs — Free tier: 1 parallel job up to 1800 mins/mo

Currently 0/1800 minutes are consumed — Purchase parallel jobs

Self-hosted — View in-progress jobs — 3 Parallel jobs

Free parallel jobs	1
Visual Studio Enterprise subscribers	2
Monthly purchases	0 Change

Public projects

Microsoft-hosted — View in-progress jobs — 10 Parallel jobs

Self-hosted — View in-progress jobs — Unlimited Parallel jobs

Describe Azure Pipelines and open-source projects

1. **How do I qualify for the free tier of public projects?**
   - The project must be a Public project and
   - Build a Public Repo

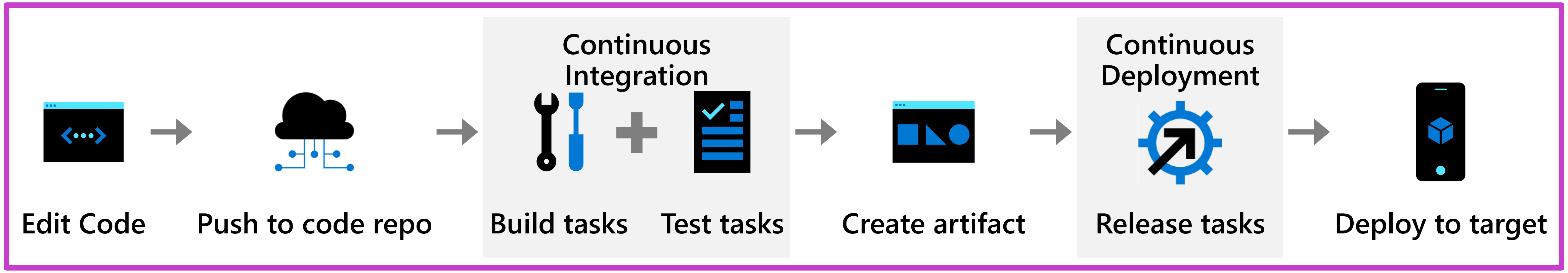
2. **Are there limits on who can use Azure Pipelines?**
   - Unlimited users – no per user cost
   - Users with Basic and Stakeholder access can author pipelines

3. **Are there any limits on the number of builds and release pipelines that I can create?**
   - No limits to the number of pipelines
   - Unlimited Self hosted Build Agents
   - 10 Free parallel jobs

4. **As a Visual Studio Enterprise subscriber, do I get additional parallel jobs for Azure Pipelines?**
   - Subscribers will get one self-hosted parallel job in each organization

Explore Azure Pipelines and Visual Designer

Configure your pipelines with the Visual Designer



Create and configure your build and release pipelines.

Push your code to your version control repository.

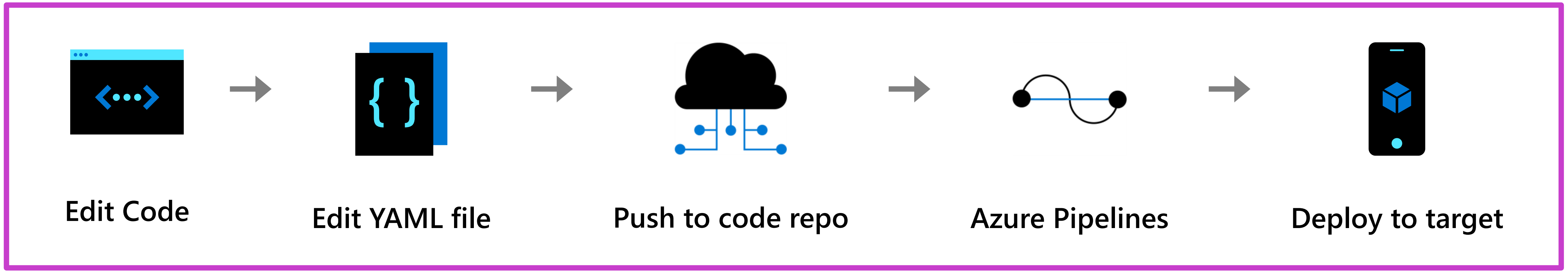
The build creates an artifact that's used by the rest of your pipeline.

Your code is now updated, built, tested, and packaged.

Often referred to as "Classic Pipelines"

Describe Azure Pipelines and YAML

Configure your pipelines in a YAML file that exists alongside your code



Configure Azure Pipelines to use your Git repo.

Edit your Azure-pipelines.yml file to define your build.

Push your code to your version control repository.

Your code is now updated, built, tested, and packaged.

While more code-oriented, defining pipelines using YAML is preferred

Module 04: Explore Continuous integration

Introduction to continuous integration

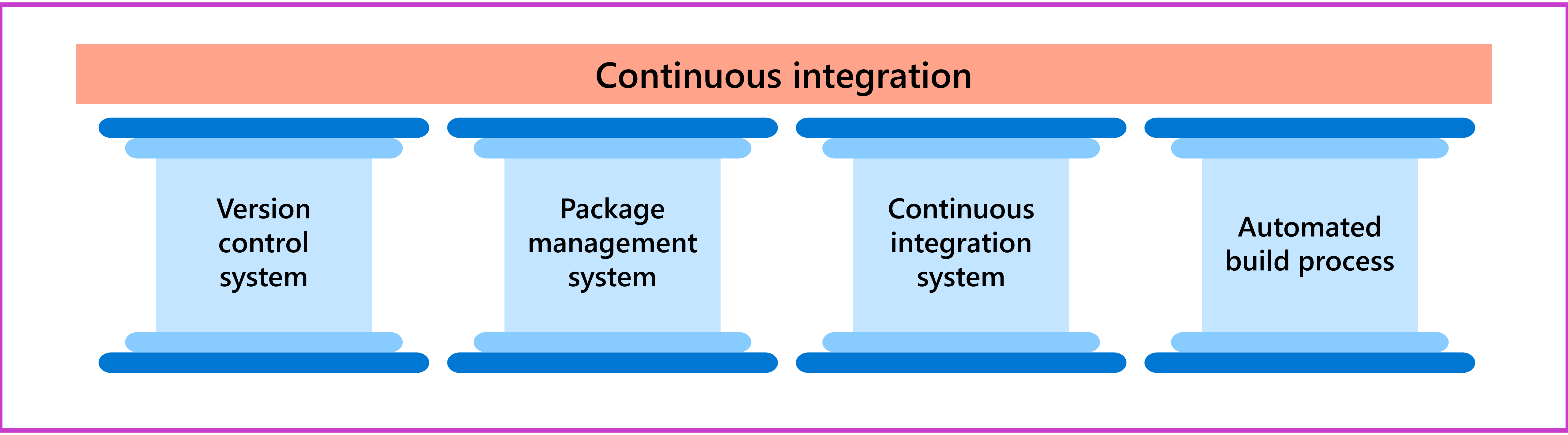
1. Continuous integration (CI) is the process of automating the build and testing of code

2. CI encourages developers to share their code and unit tests by merging their changes into the shared version control repository

3. When a change is detected, it triggers an automated build system. The code is built using a build definition. Developers respond to any issues or bugs

4. CI keeps the main branch clean ensuring bugs are caught earlier in the development cycle, which makes them less expensive to fix

Learn the four pillars of continuous integration



A **version control system** manages changes to your source code over time.

A **package management system** is used to install, uninstall, and manage software packages.

A **continuous integration system** merges all developer working copies to a shared mainline several times a day.

An **automated build process** creates a software build including compiling, packaging, and running automated tests.

Explore benefits of continuous integration

1. Improving code quality based on rapid feedback
2. Triggering automated testing for every code change
3. Reducing build times for rapid feedback and early detection of problems (risk reduction)
4. Better management of technical debt and code analysis
5. Reducing long, difficult, and bug-inducing merges
6. Increasing confidence in codebase health long before production deployment
7. Rapid feedback to the developer

Discussion: Continuous integration

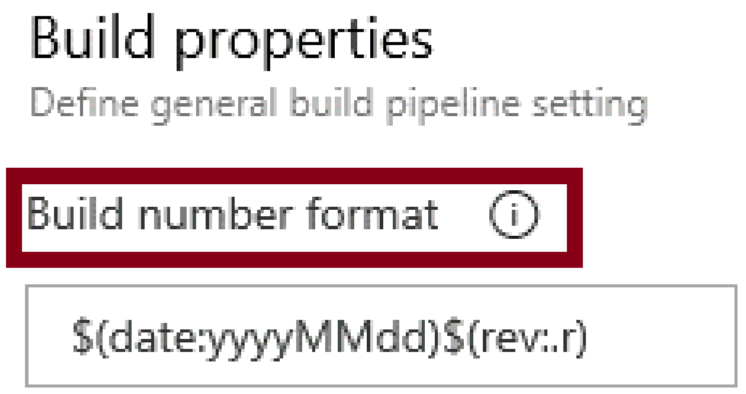
1. Have you tried to implement continuous integration in your organization?

2. If you were successful, what lessons did you learn?

3. If you were not successful, what were the challenges?

Describe build properties: Build number formatting and build status

Build number formatting

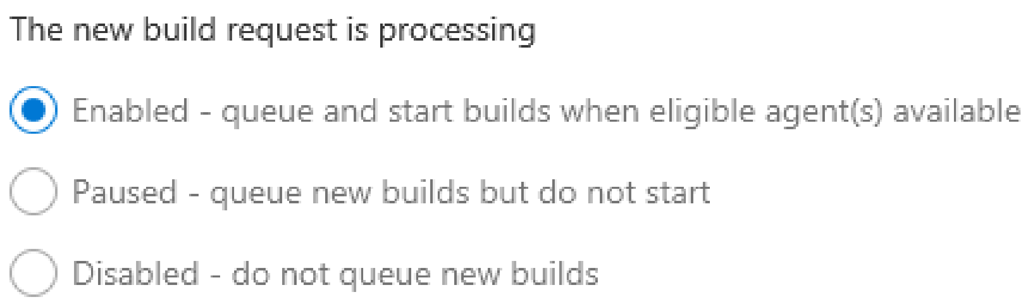

Build properties

Define general build pipeline setting

Build number format

$(date:yyyyMMdd)$(rev:.r)

Build status (enabled, paused, disabled)


The new build request is processing

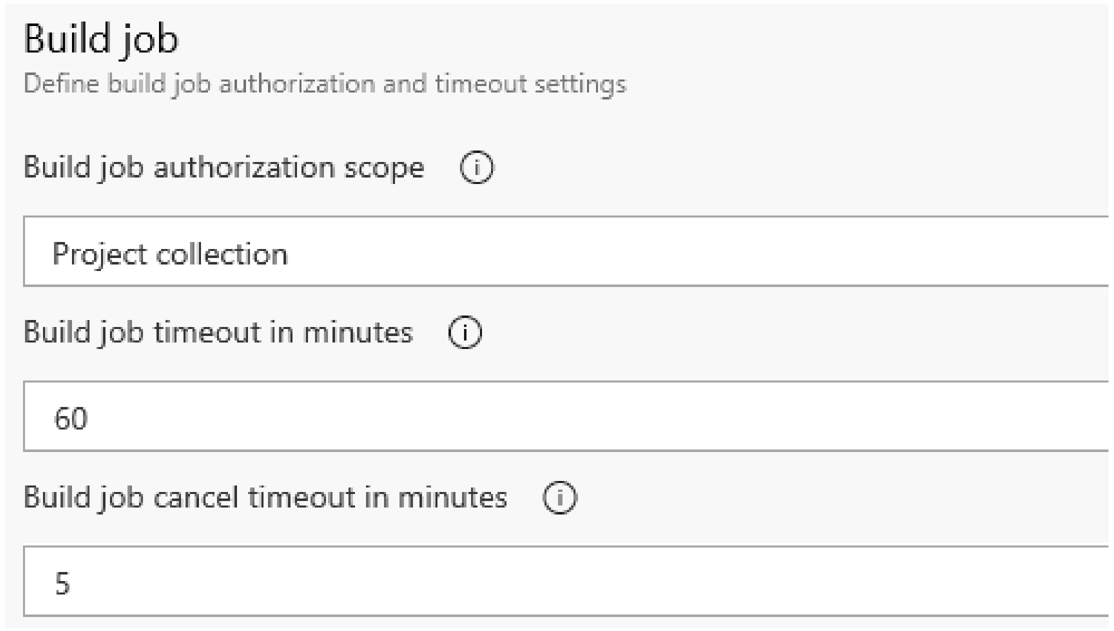
[✓ Enabled - queue and start builds when eligible agent(s) available]

Paused - queue new builds but do not start

Disabled - do not queue new builds

Describe build properties: Authorization and timeouts, and badges

Authorization and timeouts (scope, job timeout, cancel job timeout)


Build job

Define build job authorization and timeout settings

Build job authorization scope: Project collection

Build job timeout in minutes: 60

Build job cancel timeout in minutes: 5

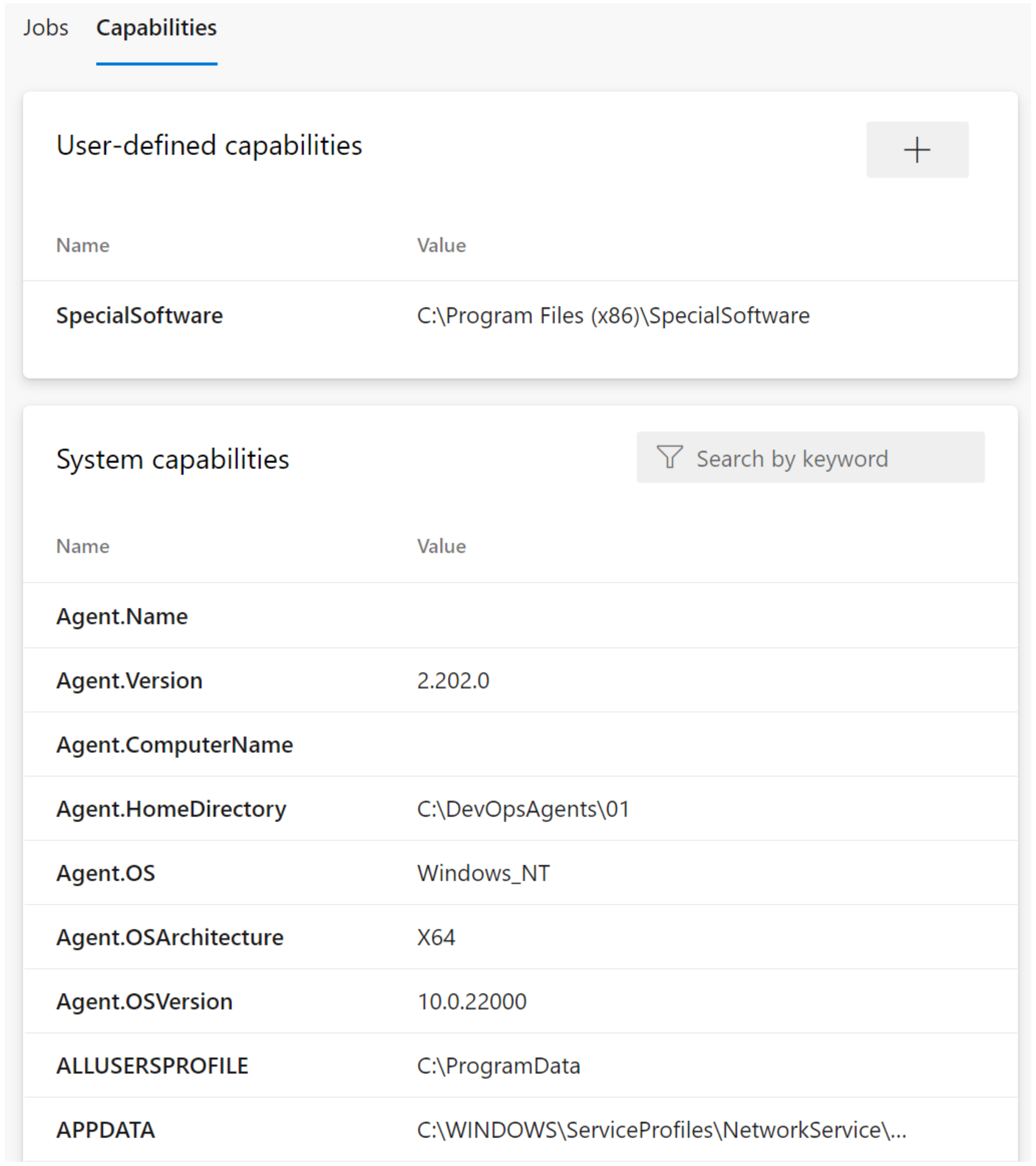
Badges

Azure Pipelines
succeeded

Module 05: Implement a pipeline strategy

Configure agent demands

- **User-defined capabilities**
- **System capabilities**
- **Agents** can have different authorization and timeout settings.



User-defined capabilities

Name	Value
SpecialSoftware	C:\Program Files (x86)\SpecialSoftware

System capabilities

Name	Value
Agent.Name	
Agent.Version	2.202.0
Agent.ComputerName	
Agent.HomeDirectory	C:\DevOpsAgents\01
Agent.OS	Windows_NT
Agent.OSArchitecture	X64
Agent.OSVersion	10.0.22000
ALLUSERSPROFILE	C:\ProgramData
APPDATA	C:\WINDOWS\ServiceProfiles\NetworkService\...

Explore multi-configuration and multi-agent

1 **Multi-configuration:** Run the same set of tasks on multiple configurations:

- Run the release once with configuration setting A on WebApp A and setting B for WebApp B
- Deploy to different geographic regions

2 **Multi-agent:** Run the same set of tasks on multiple agents using the specified number of agents:

- Deploy same bits to a farm of servers

Implement multi-job builds

Adding multiple jobs to a pipeline lets you:

1. Break your pipeline into sections that need different agent pools, or self-hosted agents
2. Publish artifacts in one job and consume them in one or more subsequent jobs
3. Build faster by running multiple jobs in parallel
4. Enable conditional execution of tasks

✓ **You can configure the number of parallel jobs**

Discussion: Build-related tooling

Azure DevOps can be integrated with a wide range of existing tooling that is used for builds or associated with builds

1. Which build-related tools do you currently use?
2. What do you like or don't like about the tools?

Explore source control types supported by Azure Pipelines

Repository types supported:

Repository Type	Azure Pipelines (YAML)	Azure Pipelines (Classic Editor)
Azure Repos Git	Yes	Yes
Azure Repos TFVC	No	Yes
Bitbucket Cloud	Yes	Yes
Other Git (Generic)	No	Yes
GitHub	Yes	Yes
GitHub Enterprise Server	Yes	Yes
Subversion	No	Yes

Module 06: Integrate with Azure Pipelines

Describe the anatomy of a pipeline

1. **Name** – Though often this is skipped (if it is skipped, a date-based name is generated automatically)
2. **Trigger** – More on triggers later, but without an explicitly trigger, there's an implicit "trigger on every commit to any path from any branch in this repo"
3. **Variables** – These are "inline" variables (more on other types of variables later)
4. **Job** – Every pipeline must have at least one job
5. **Pool** – You configure which pool (queue) the job must run on
6. **Checkout** – The "checkout: Self" tells the job which repository (or repositories if there are multiple checkouts) to checkout for this job
7. **Steps** – The actual tasks that need to be executed: in this case a "script" task (script is an alias) that can run inline scripts

Understand the pipeline structure

1. **Pipeline –** One or more stages that describe a CI/CD process
2. **Stage –** Collection of related jobs
3. **Job –** Collection of steps run by an agent on a server
4. **Step –** Linear sequence of operations that make up a job
5. **Tasks –** Building blocks of a pipeline

Detail YAML templates

Azure Pipelines supports 4 kinds of templates:

1. Stage
2. Job
3. Step
4. Variable

Explore YAML resources

Pipelines

Repositories

Containers

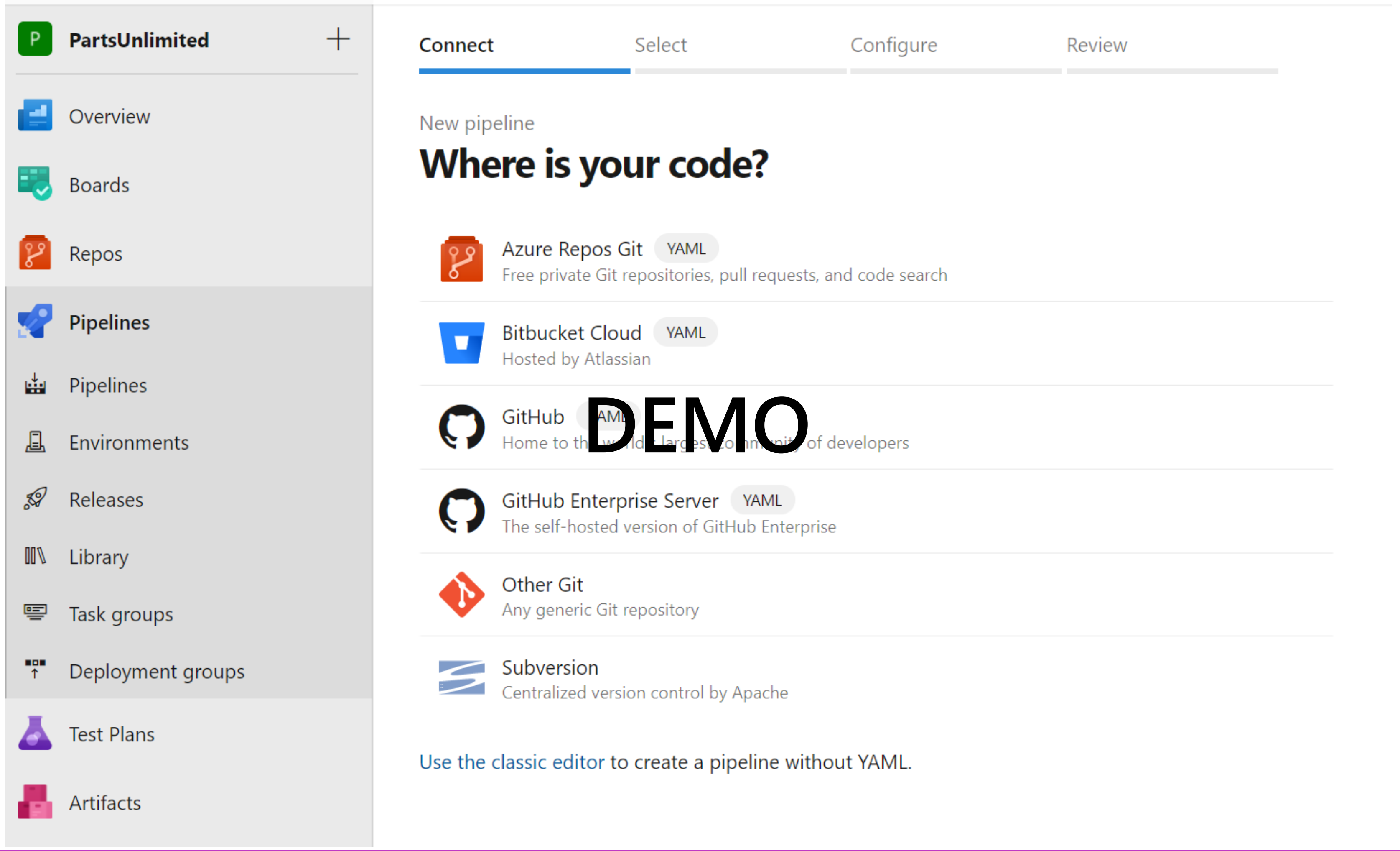
```
resources:
    pipelines: [ pipeline ]
    repositories: [ repository ]
    containers: [ container ]
```

```
resources:
  pipelines:
  - pipeline: MyAppA
    source: MyCIPipelineA
  - pipeline: MyAppB
    source: MyCIPipelineB
    trigger: true
  - pipeline: MyAppC
    source: MyCIPipelineC
    branch: releases/M174
    version: 20230709.2
    trigger:
      branches:
        include:
          - main
          - releases/*
        exclude:
          - users/*
```

Use multiple repositories in your pipeline

1. Common for utilities used across more than one pipeline
2. Add additional checkout steps
3. Was not supported in earlier versions and artifacts were used as a workaround
4. Some pipelines might not need any repository
5. Build and validate pull requests and commits to your **GitHub** repository with Azure Pipelines

GitHub repository with Azure Pipelines



New pipeline

Where is your code?

Azure Repos Git YAML
Free private Git repositories, pull requests, and code search

Bitbucket Cloud YAML
Hosted by Atlassian

GitHub YAML
Home to the world's largest community of developers

GitHub Enterprise Server YAML
The self-hosted version of GitHub Enterprise

Other Git
Any generic Git repository

Subversion
Centralized version control by Apache

Use the classic editor to create a pipeline without YAML.

DEMO

Module 07: Introduction to GitHub Actions

What are Actions?

1. Automations within the GitHub environment
2. Often used to build CI/CD implementations
3. Based on YAML files living within GitHub repositories
4. Executed on GitHub or self-hosted runners
5. Large number of existing actions in the GitHub Marketplace

Explore Actions flow

Events trigger workflows:

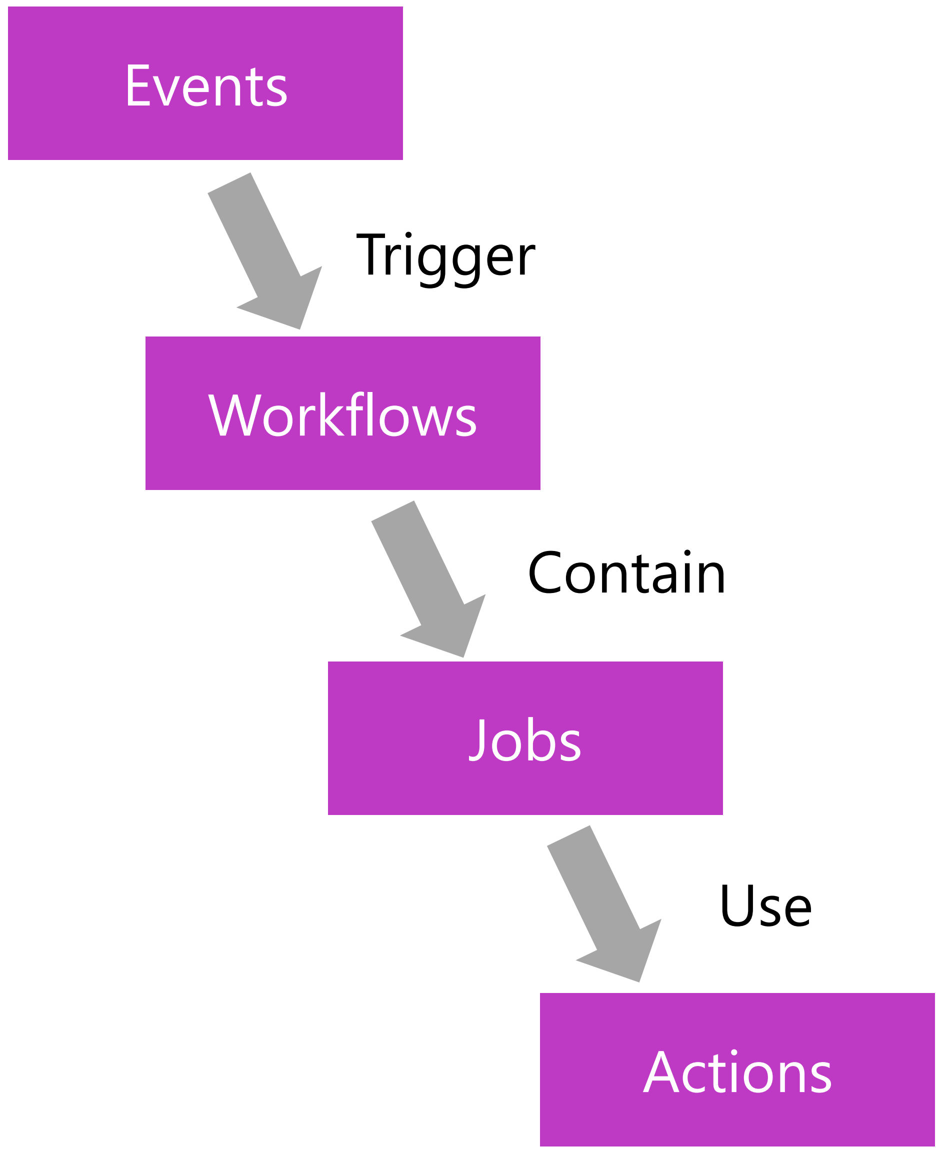
- Schedule, code, etc.

Workflows contain jobs:

- May contain multiples

Jobs use actions:

- Configured within steps



Understand Workflows

Define the required automation:

- Events, Jobs
- Written in YAML
- Stored in **.github/workflows**
- Starter workflows available

```yaml
# .github/workflows/build.yml
name: Node Build

on: push

jobs:
  mainbuild:
    runs-on: ${{ matrix.os }}
    strategy:
      matrix:
        node-version: [12.x]
        os: [windows-latest]

    steps:
    - uses: actions/checkout@v1
    - name: Run node.js on latest Windows
      uses: actions/setup-node@v1
      with:
        node-version: ${{ matrix.node-version }}
    - name: Install NPM and build
      run: |
        npm ci
        npm run build
```

Describe standard workflow syntax elements

1. **Name:** The name of the workflow (optional but recommended)
2. **On:** The event that triggers the workflow
3. **Jobs:** The list of jobs to be executed
4. **Runs-on:** The target runner
5. **Steps:** The list of steps to execute
6. **Uses:** A predefined action to retrieve
7. **Run:** Tells the job to execute a command on the runner

Explore Events

Defined by the on clause

- Scheduled events
- Code events
- Manual events
- Webhook events
- External events

```
on:
  schedule:
      - cron: '0 8-17 * * 1-5'
```

```
on:
  pull_request
```

```
on:
  [push, pull_request]
```

```
on:
  pull_request:
      branches:
          - develop
```

```
on:
  gollum
```

Explore Jobs

```yaml
jobs:
  startup:
    runs-on: ubuntu-latest
    steps:
      - run: ./setup_server_configuration.sh
  build:
    steps:
      - run: ./build_new_server.sh
```

```yaml
jobs:
  startup:
    runs-on: ubuntu-latest
    steps:
      - run: ./setup_server_configuration.sh
  build:
    needs: startup
    steps:
      - run: ./build_new_server.sh
```

Workflows contain one or more jobs. Jobs contain a set of steps to execute in order.

Jobs run in parallel by default but can be configured with dependencies.

Explore Runners

Job steps execute on runners:

- Shell scripts
- Actions

For JavaScript code, Node.js hosted runners:

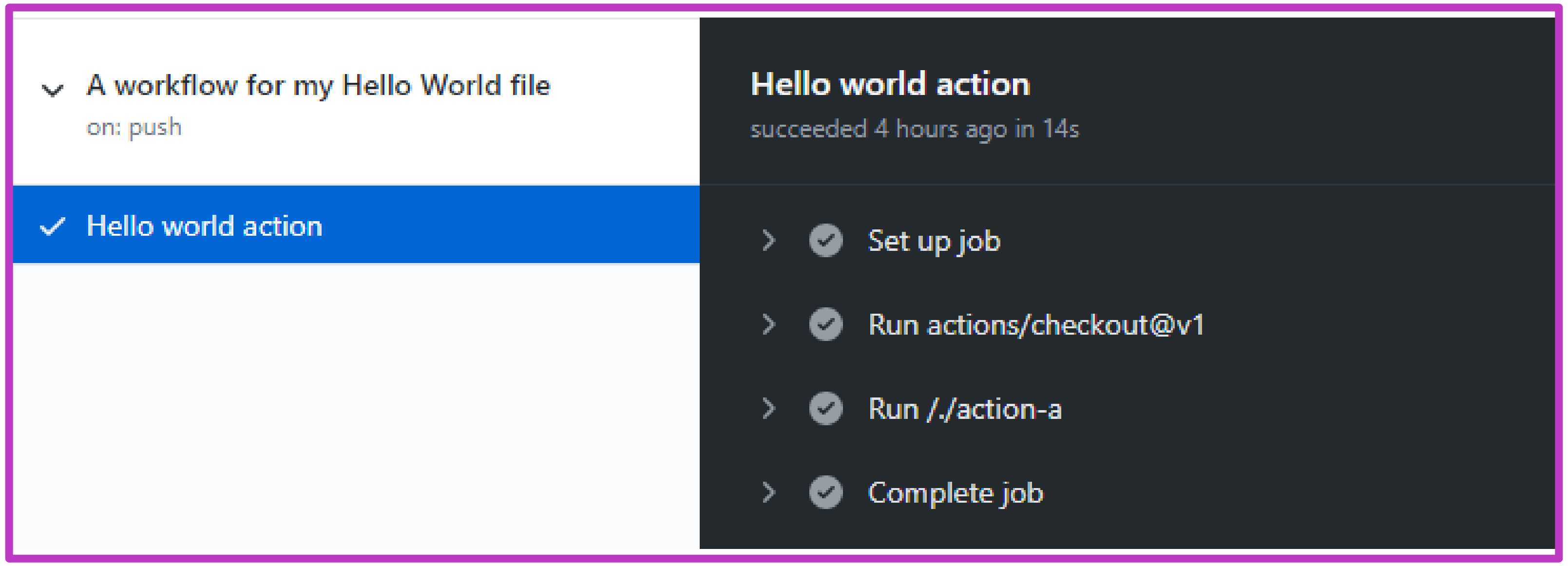
- Windows
- MacOS
- Linux

For other languages or options:

- Docker containers on Linux
- Self-hosted – but do not use for public repos

Console output from actions

- Console output from actions is available directly in the GitHub UI



A workflow for my Hello World file
on: push

✓ Hello world action

Hello world action
succeeded 4 hours ago in 14s

- Set up job
- Run actions/checkout@v1
- Run /./action-a
- Complete job

Examine release and test an action

Specific releases of actions can be requested:

- Using tags
- Using branches
- Using SHA hashes

```yaml
steps:
  - uses: actions/install-timer@v2.0.1
```

```yaml
steps:
  - uses: actions/install-timer@develop
```

```yaml
steps:
  - uses: actions/install-timer@327239021f7cc39fe7327647b213799853a9e
```

Testing an action

DEMO

Module 08: Learn continuous integration with GitHub Actions

Describe continuous integration with actions

Example workflow shown:

- Executes when code is pushed
- Runs on latest ubuntu
- Uses Node.js version 10

Steps:

- Check out the code
- Set up dotnet
- Build the project

```yaml
name: dotnet Build

on: [push]

jobs:
  build:
    runs-on: ubuntu-latest
    strategy:
      matrix:
        node-version: [10.x]

    steps:
    - uses: actions/checkout@main
    - uses: actions/setup-node@v1
      with:
        dotnet-version: '3.1.x'
    - run: dotnet build eShopOnWeb
```

Examine environment variables

CI workflows usually need environment variables

- GitHub default variables (GITHUB_ prefix)
- Custom variables

```yaml
jobs:
  verify-connection:
     steps:
        - name: Verify Connection to SQL Server
          run: node testconnection.js
          env:
            PROJECT_SERVER: PH202323V
            PROJECT_DATABASE: HAMain
```

Share artifacts between jobs

CI workflows need to be able to pass artifacts between jobs. Standard actions:

- Upload-artifact
- Download-artifact

Can configure retention

```yaml
uses: actions/upload-artifact
with:
  name: harness-build-log
  path: bin/output/logs/harness.log
```

```yaml
uses: actions/upload-artifact
with:
  name: harness-build-logs
  path: bin/output/logs/
```

```yaml
uses: actions/upload-artifact
with:
  name: harness-build-logs
  path: bin/output/logs/harness[ab]?/*
```

```yaml
uses: actions/upload-artifact
with:
  name: harness-build-log
  path: |
      bin/output/logs/harness.log
      bin/output/logs/harnessbuild.txt
```

Examine Workflow badges

Show status of a workflow within a repository

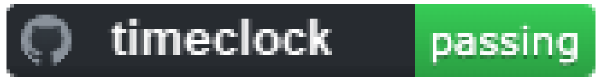
Typically added to README.md

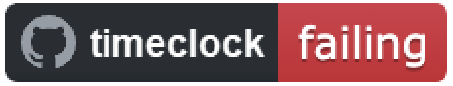
Can be branch specific

Added via URLs

https://github.com/**accountname/repositoryname**/workflows/**workflowname**/badge.svg

Spaces in names must be encoded as %20



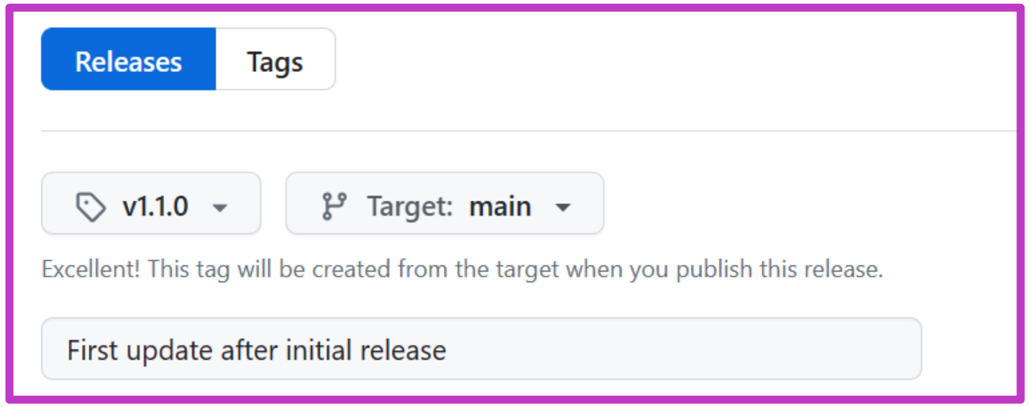


Describe best practices for creating actions

1. Create chainable actions – avoid monolithic actions
2. Version your actions like other code
3. Provide a **latest** label
4. Add appropriate documentation: like README.md
5. Add detailed **action.yml** metadata
6. Consider contributing to the marketplace

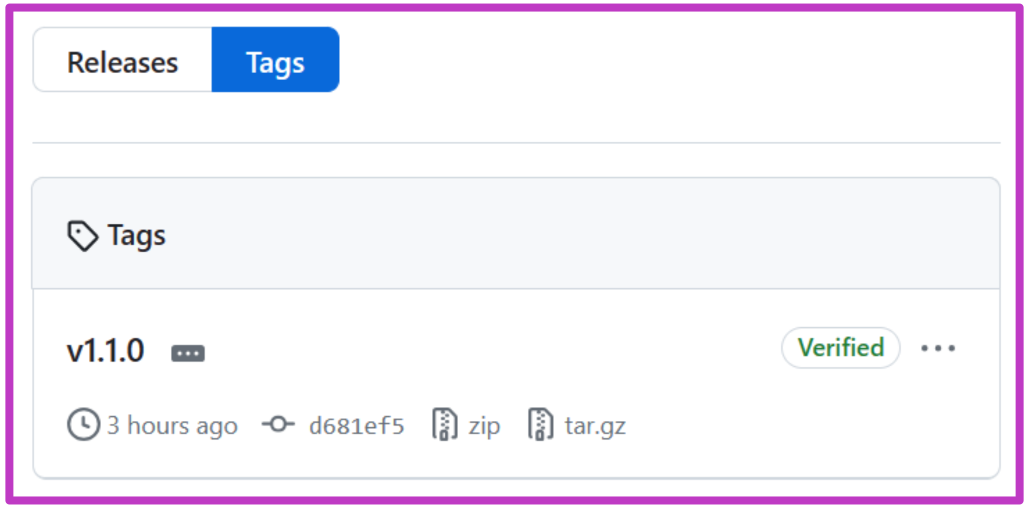
Mark releases with Git tags

- Releases are based on Git tags
- Mark a specific point in repository history
- Tags are viewed in the repository history



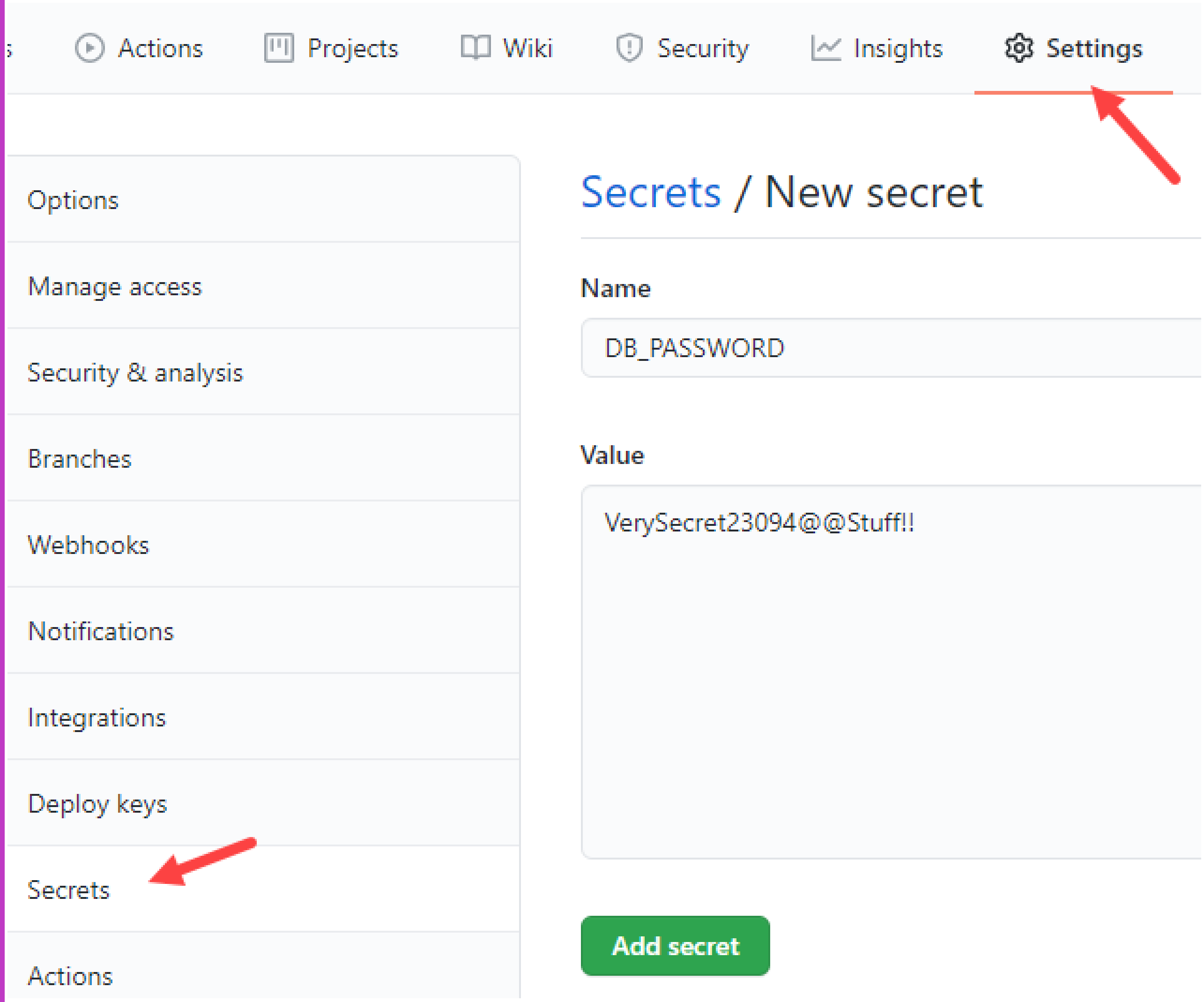
Excellent! This tag will be created from the target when you publish this release.

First update after initial release



Create encrypted secrets

- Like environment variable but encrypted
- Created at repository or organization level
- Created/assigned in the GitHub UI



Secrets / New secret

Name

DB_PASSWORD

Value

VerySecret23094@@Stuff!!

Use secrets in a workflow

- Secrets not automatically passed to runners
- Can be passed as inputs or as environment variables
- Avoid passing secrets in command-line arguments

```
steps: actions/upload-artifact
  - name: Test Database Connectivity
    with:
      db_username: ${{ secrets.DBUserName }}
      db_password: ${{ secrets.DBPassword }}
```

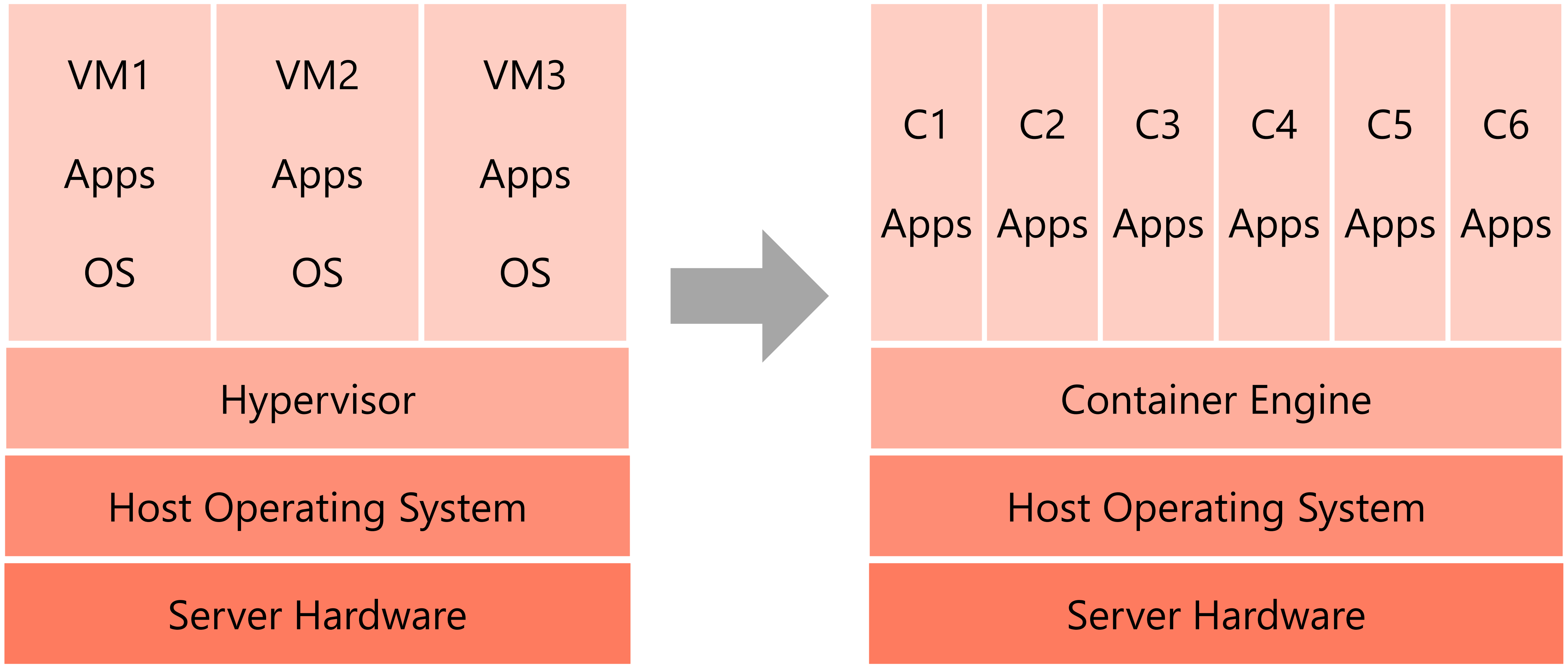
```
steps:
  - shell: pwsh
    env:
      DB_PASSWORD: ${{ secrets. DBPassword }}
    run: |
      db_test "$env.DB_PASSWORD"
```

Module 09: Design a container build strategy

Why Containers?

1. **Portable –** Run containers wherever Docker is supported
2. **Consistent –** Know that developers are working with identical code
3. **Lightweight –** Far less disk, CPU, and memory than VMs
4. **Efficient –** Fast deployment, booting, patching, updates

Examine structure of containers



Discussion: Containers versus Virtual Machines

In your development environment:

1. Do you currently use virtualization of any type?

2. Do you prefer to deploy containers or virtual machines?

Work with Docker containers

1. **Docker build –** Create the image
2. **Docker pull –** Retrieve an image from a container registry
3. **Docker run –** Run the image to create a container instance (will auto pull if not already pulled)

Understand Dockerfile core concepts

```
FROM ubuntu
LABEL maintainer="johndoe@contoso.com"
ADD appsetup /
RUN /bin/bash -c 'source $HOME/.bashrc; \
echo $HOME'
CMD ["echo", "Hello World from within the container"]
```

✓ Dockerfiles are text files that contain the commands needed by **docker build** to assemble an image

Multi-stage Dockerfiles

Multi-stage builds give the benefits of the builder pattern without the hassle of maintaining separate files

The code progresses from one stage to the next

```
FROM mcr.microsoft.com/dotnet/core/aspnetcore:3.1
AS base
WORKDIR /app
EXPOSE 80
EXPOSE 443

FROM mcr.microsoft.com/dotnet/core/sdk:3.1 AS
build
WORKDIR /src
COPY ["WebApplication1.csproj", ""]
RUN dotnet restore "./WebApplication1.csproj"
COPY . .
WORKDIR "/src/."
RUN dotnet build "WebApplication1.csproj" -c
Release -o /app/build

FROM build AS publish
RUN dotnet publish "WebApplication1.csproj" -c
Release -o /app/publish

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "WebApplication1.dll"]
```

Considerations for multiple stage builds

1. Adopt container modularity
2. Avoid unnecessary packages
3. Choose an appropriate base
4. Avoid including application data

Explore Azure container-related services

1. **Azure Container Instances** let you focus on creating your applications rather than provisioning and management of the infrastructure
2. **Azure Kubernetes Service** is the de facto standard for container orchestration
3. **Azure Container Registry** lets you store and manage container images in a central registry
4. **Azure Container Apps** to build and deploy modern apps and microservices using serverless containers
5. **Azure App Service** provides a managed service for both Windows and Linux based web applications

Labs

Lab: Configuring Agent Pools and Understanding Pipeline Styles

Lab overview:

In this lab, you will step through the process of converting a classic pipeline into a YAML-based one and running it first by using a Microsoft-hosted agent and then performing the equivalent task by using a self-hosted agent.

Objectives:

- Implement YAML-based pipelines
- Implement self-hosted agents

Duration:

Lab: Enabling continuous integration with Azure Pipelines

Lab overview:

In this lab, you will learn how to configure continuous integration (CI) and continuous deployment (CD) for your applications using Build and Release in Azure Pipelines.

Objectives:

- Create a basic build pipeline from a template
- Track and review a build
- Invoke a continuous integration build

Duration:

60 minutes

Lab: Implementing GitHub Actions for CI/CD

Lab overview:

In this lab, you will learn how to implement a GitHub Action workflow that deploys an Azure web app by using DevOps Starter.

Objectives:

- Implement a GitHub Action workflow by using DevOps Starter
- Explain the basic characteristics of GitHub Action workflows

Duration:

30
minutes

Lab: Deploying Docker containers to Azure App Service web apps

Lab overview:

In this lab, you will learn how to use an Azure DevOps CI/CD pipeline to build a custom Docker image, push it to Azure Container Registry, and deploy it as a container to Azure App Service.

Objectives:

- Build a custom Docker image by using a Microsoft hosted Linux agent
- Push an image to Azure Container Registry
- Deploy a Docker image as a container to Azure App Service by using Azure DevOps

Duration:

60 minutes

Learning Path review and takeaways

What did you learn?

1. Explain the role of Azure Pipelines and its components
2. Implement a container strategy including how containers are different from virtual machines
3. Configure agents for use in Azure Pipelines
4. Explain why continuous integration matters

What did you learn? (continued)

5 Implement continuous integration using Azure DevOps

6 Create and work with GitHub Actions and workflows

7 Implement Continuous Integration with GitHub Actions

Learning Path review questions

1. What are some advantages of Azure Pipelines?
2. What is a pipeline, and why is it used?
3. What is the line continuation character in Dockerfiles?
4. What are the two types of agents and how are they different?
5. What is an agent pool, and why would you use it?
6. What are two ways to configure your Azure Pipelines?

Learning Path review questions (continued)

7. Name the four pillars of continuous integration.

8. You want to take your build server offline to make a configuration change. You want it to complete any build that it is currently processing, but you want to queue any new build requests. What should you do?

9. You want to set a maximum time that builds should not run for more than 5 minutes. What configuration change should you make?

Learning Path review questions (continued)

10 True or False: Self-hosted runners should be used with public repos.

11 Database passwords that are needed in a CI pipeline should be stored where?

12 The metadata for an action is held in which file?

13 How can the status of a workflow be shown in a repository ?

Microsoft