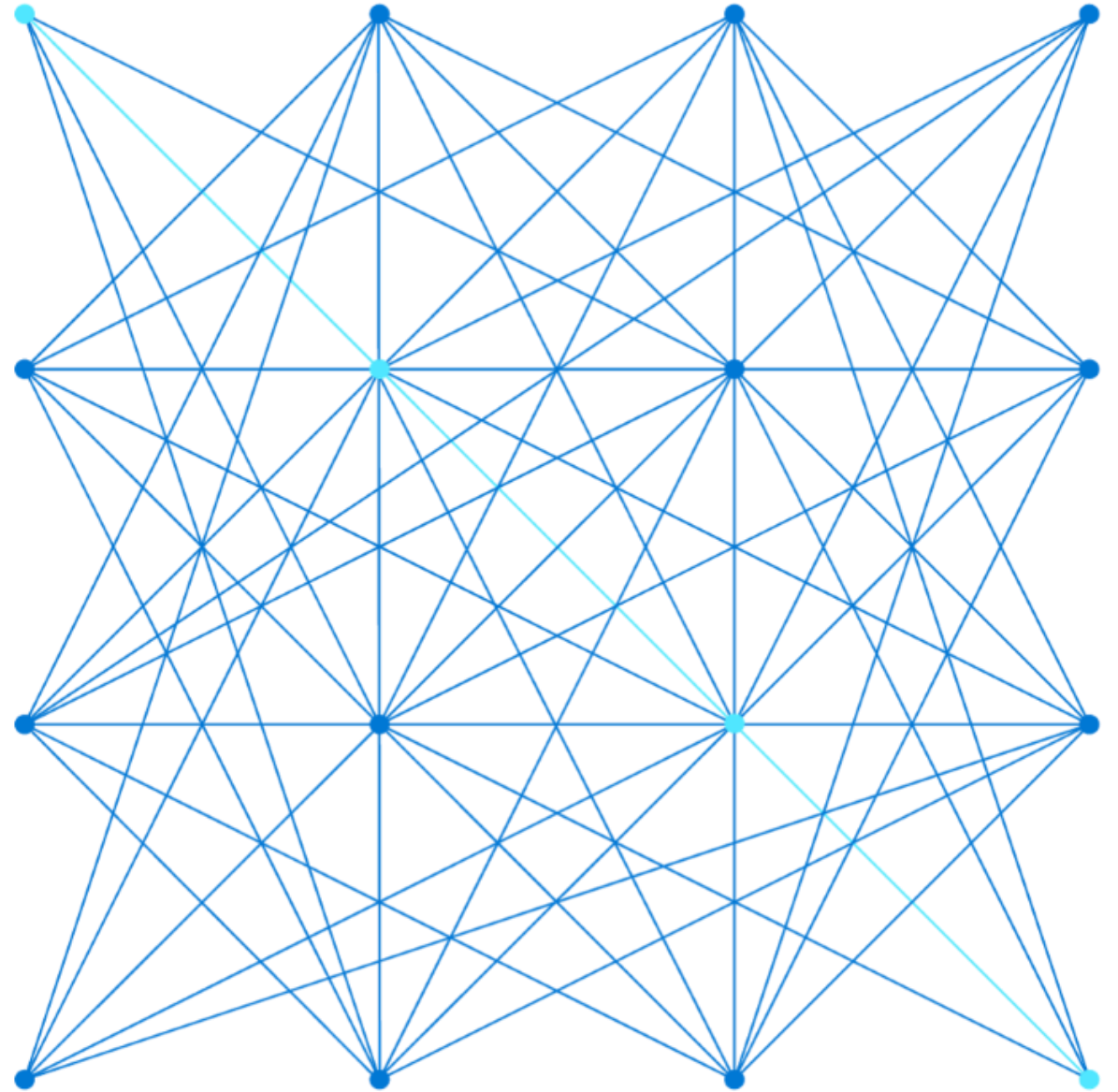


AZ-400.00

Module 6: Implementing Continuous Integration using Azure Pipelines



Lesson 01: Module overview



Module overview



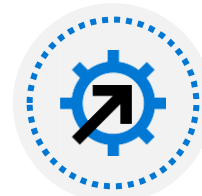
Lesson 1: Module overview



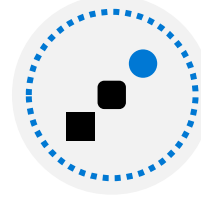
Lesson 2: Continuous integration overview



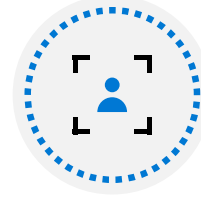
Lesson 3: Implementing a build strategy



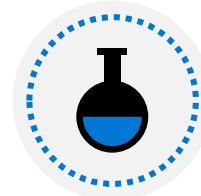
Lesson 4: Integration with Azure Pipelines



Lesson 5: Integrating external source control with Azure Pipelines



Lesson 6: Set up self-hosted agents



Lesson 7: Labs



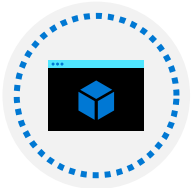
Lesson 8: Module review and takeaways

Learning objectives

After completing this module, students will be able to:



Explain why continuous integration matters



Implement continuous integration using Azure DevOps

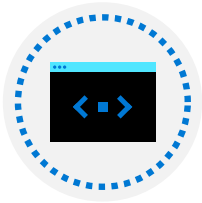
Lesson 02: Continuous integration overview



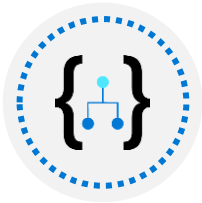
Introduction to continuous integration



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



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

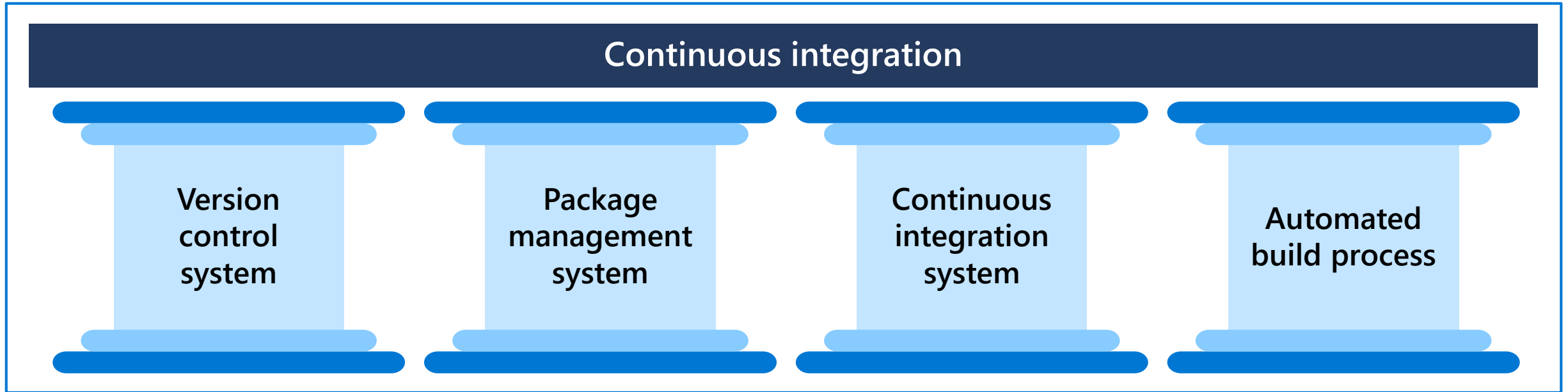


When a change is detected, it triggers an automated build system. The code is built using a build definition. Developers respond to any issues or bugs



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

The four pillars of continuous integration



A **version control system** manages changes to your source code over time. (e.g. GIT)

A **package management system** is used to install, uninstall, and manage software packages. (e.g. npm)

A **continuous integration system** merges all developer working copies to a shared mainline several times a day. (e.g. AzureDevOps, Jenkins)

An **automated build process** creates a software build including compiling, packaging, and running automated tests. (e.g. Apache Ant, Gradle)

Benefits of continuous integration



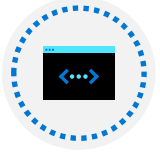
Improving code quality based on rapid feedback



Triggering automated testing for every code change



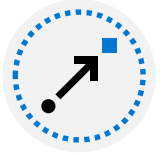
Reducing build times for rapid feedback and early detection of problems (risk reduction)



Better management of technical debt and code analysis



Reducing long, difficult, and bug-inducing merges



Increasing confidence in codebase health long before production deployment



Rapid feedback to the developer

Build number formatting and build status

(#mw-CLASSIC pipeline interface)

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

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



Lesson 03: Implementing a build strategy



Configuring agent demands

(#mw – this is the old UI)

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

[Requests](#) [Capabilities](#)

USER CAPABILITIES

Shows information about user-defined capabilities supported by this host

+

Add capability

Save changes

Undo changes

SYSTEM CAPABILITIES

Shows information about the capabilities provided by this host

Capability name	Capability value
Agent.ComputerName	GREGP50
Agent.HomeDirectory	C:\agent
Agent.Name	GREGP50
Agent.OS	Windows_NT
Agent.OSArchitecture	X64
Agent.OSVersion	10.0.17134
Agent.Version	2.141.2
ALLUSERSPROFILE	C:\ProgramData
APPDATA	C:\Users\Greg\AppData\Roaming
AzurePS	5.7.0
bower	C:\Users\Greg\AppData\Roaming\npm\bower.cmd

(#mw new way – Org settings -> Agent Pools -> (e.g. AzurePipelines) -> Agents -> Capabilities)

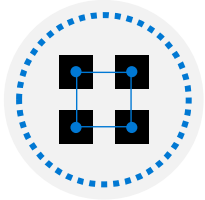
#mw. [remember jobs?](#)

#mw. E.g. [classic](#)

#mw: E.g. [YAML](#)

Implementing multi-agent builds

Adding multiple **jobs** to a pipeline lets you:



Break your pipeline into sections that need different agent pools, or self-hosted agents



Publish artifacts in one job and consume them in one or more subsequent jobs



Build faster by running multiple jobs in parallel



Enable conditional execution of tasks



You can configure the number of parallel jobs (organization settings->Parallel Jobs)

Lesson 04: Integrate with Azure Pipelines



Anatomy of a pipeline



Name – Though often this is skipped (if it is skipped, a date-based name is generated automatically)



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"



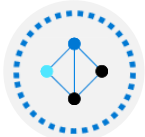
Variables – These are "inline" variables (more on other types of variables later)



Job – Every pipeline must have at least one job



Pool – You configure which pool (queue) the job must run on

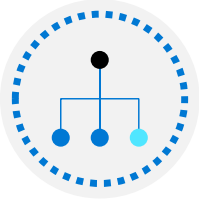


Checkout – The "checkout: self" tells the job which repository (or repositories if there are multiple checkouts) to checkout for this job

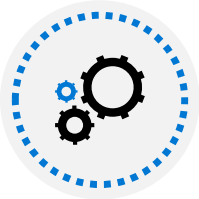


Steps – The actual tasks that need to be executed: in this case a "script" task (script is an alias) that can run inline scripts

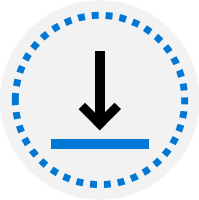
Pipeline structure



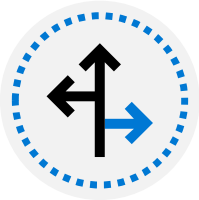
Pipeline – One or more stages that describe a CI/CD process (e.g. [helloworld](#))



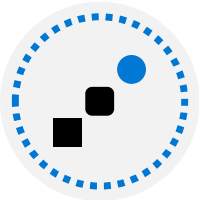
Stage – Collection of related jobs (e.g. [stages](#))



Job – Collection of steps run by an agent on a server (e.g. [parallel](#), [dependent](#))



Step – Linear sequence of operations that make up a job



Tasks – Building blocks of a pipeline

(#mw) Hello world (from your notes)

#mw-d- Pipeline Build Name

```
name: 1.0$(Rev:.r)

# simplified trigger (implied branch)
trigger:
- master

# equivalent trigger
# trigger:
#   branches:
#     include:
#       - master

variables:
  name: martin

pool:
  vmImage: ubuntu-latest

jobs:
- job: helloworld
  steps:
  - script: echo "Hello, $(name)"
```

It's best to add a name otherwise you'll just get an incremental number for each run of the pipeline (1,2,3....). This example returns a format of 1.0.x.

This indicates to trigger the pipeline if ever a change is merged into master

Variables can be used later in the template

Which agent to use

A job is a series of steps to execute

The steps contain the tasks to perform

Steps can have one or more tasks

(#mw) Run jobs in Parallel (from your notes)

```
jobs:
- job: A
  steps:
    # steps omitted for brevity

- job: B
  dependsOn: none
  steps:
    # steps omitted for brevity
```

(#mw) Run dependent jobs (from your notes)

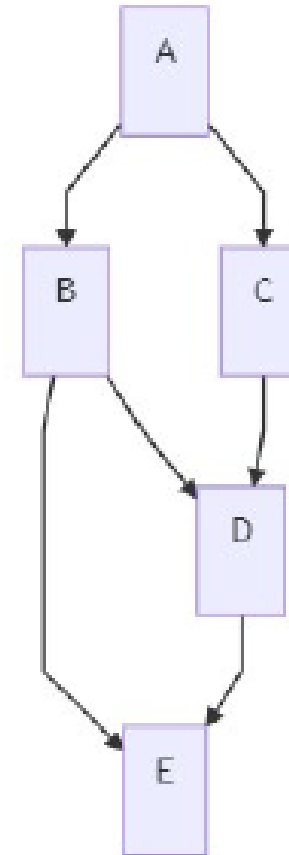
```
jobs:
- job: A
  steps:
  - script: echo 'job A'

- job: B
  dependsOn: A
  steps:
  - script: echo 'job B'

- job: C
  dependsOn: A
  steps:
  - script: echo 'job C'

- job: D
  dependsOn:
  - B
  - C
  steps:
  - script: echo 'job D'

- job: E
  dependsOn:
  - B
  - D
  steps:
  - script: echo 'job E'
```



(#mw) Stages (from your notes)

```
stages:
- stage: Build
  jobs:
  - job: BuildJob
    steps:
    - script: echo Building!
- stage: Test
  jobs:
  - job: TestOnWindows
    steps:
    - script: echo Testing on Windows!
  - job: TestOnLinux
    steps:
    - script: echo Testing on Linux!
- stage: Deploy
  jobs:
  - job: Deploy
    steps:
    - script: echo Deploying the code!
```

The middle stage runs 2 jobs in parallel



Templates

(#mw) You can **export** reusable sections of your pipeline to a **separate file**. ([Example](#))

Azure Pipelines supports 4 kinds of templates:



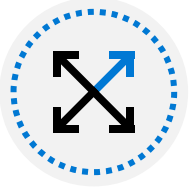
Stage

Job

Step

Variable

YAML resources



A resource is any external service that is consumed as part of your pipeline.

E.g. Secure files, service connections, artifacts



(#mw - A resource is anything used by a pipeline that lives outside the pipeline. E.g. Secure files, service connections, artifacts)

YAML resources example

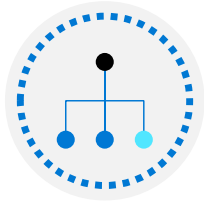
Requirement: I need to call another pipeline in the same project

```
resources:  
  pipelines:  
    - pipeline: SmartHotel-resource # identifier for the resource  
      source: SmartHotel-CI # name of pipeline that produces artifact
```

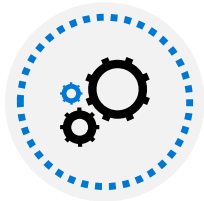
Requirement: I need to use a variable group in my pipeline

```
variables:  
  - group: my-variable-group
```

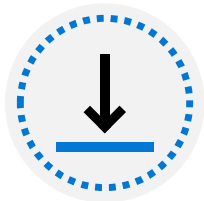
Using **multiple repositories** in your **pipeline**



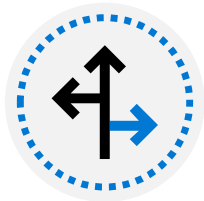
Common for utilities used across more than one pipeline



Add additional checkout steps



Was not supported in earlier versions and artifacts were used as a workaround



Some pipelines might not need any repository

Lesson 05: Integrating external source control with Azure Pipelines



Source control types supported by Azure Pipelines

Repository types supported:

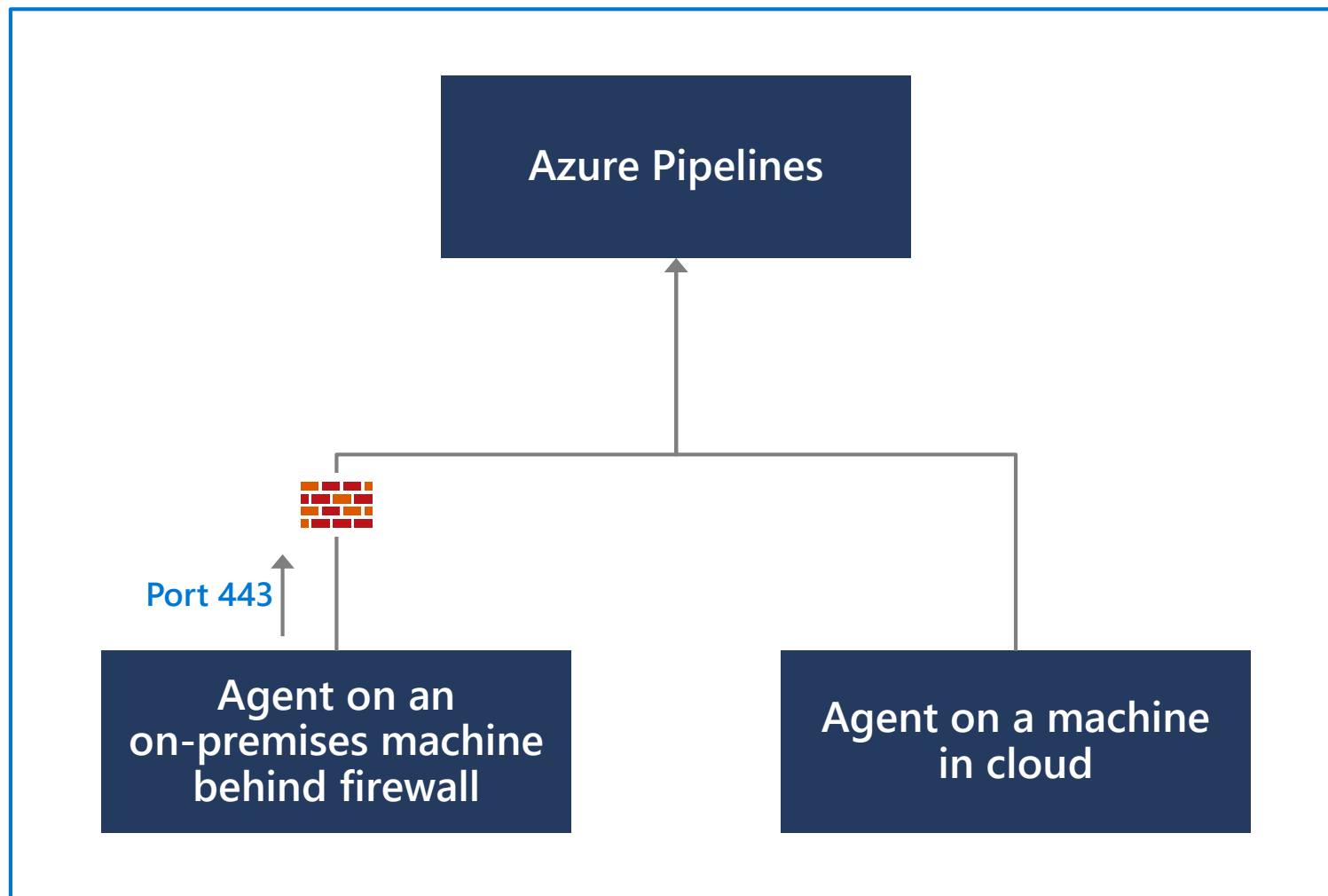
Repository Type	Azure Pipelines (YAML)	Azure Pipelines (Classic Editor)
Azure Repos Git	Yes	Yes
Azure Repos TFVC	No	Yes
Bitbucket Cloud	Yes	Yes
Other Git (Generic)	No	Yes
GitHub	Yes	Yes
GitHub Enterprise Server	Yes	Yes
Subversion	No	Yes

Lesson 06: Set up self-hosted agents



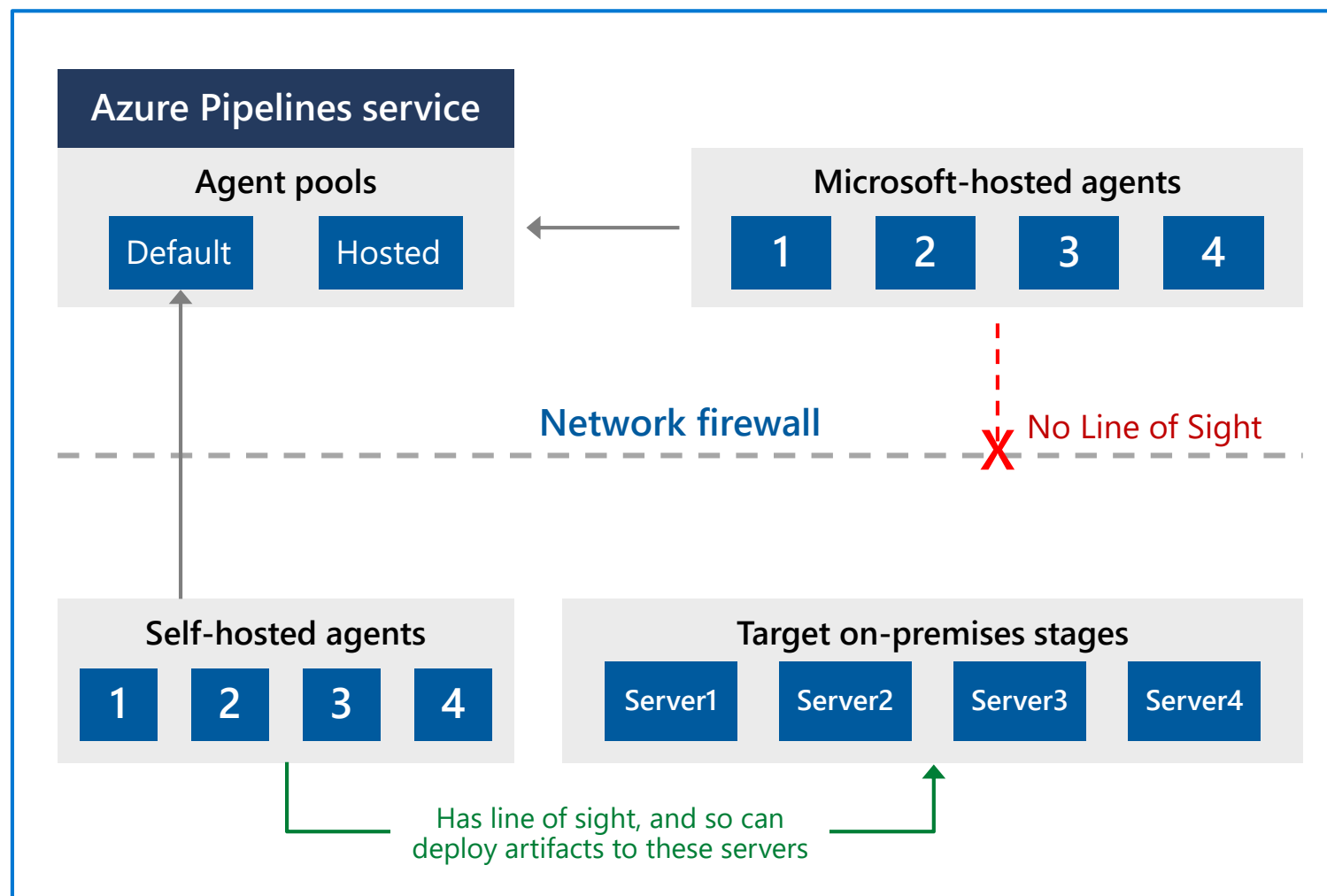
Communication with Azure Pipelines

- 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.

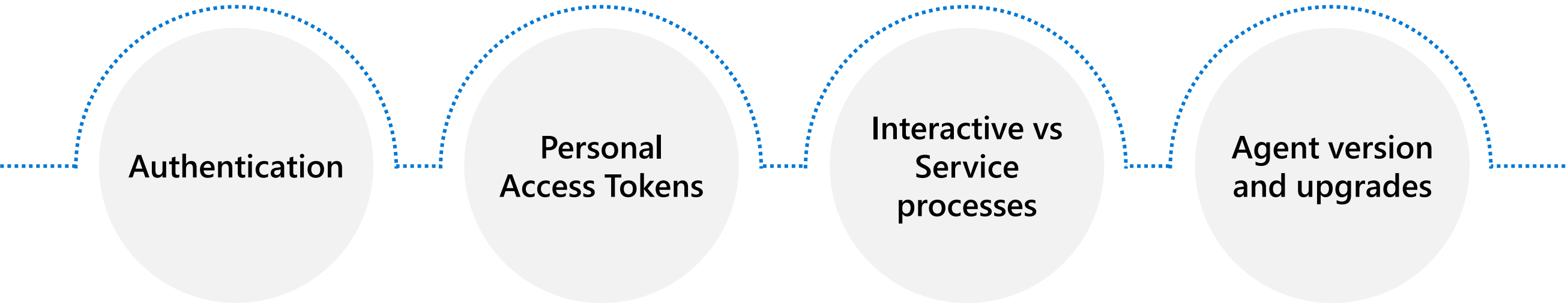


Communication to deploy to target servers

- 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.

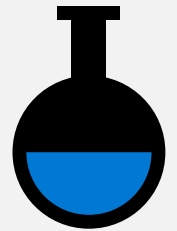


Other considerations (for self-hosted agents)



Lesson 07: Labs

Only 6a
Can do 6b outside class time



Lab: Enabling continuous integration with Azure Pipelines

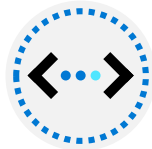


Lab overview:

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:



Lab: Integrating external source control with Azure Pipelines

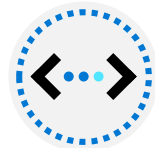


Lab overview:

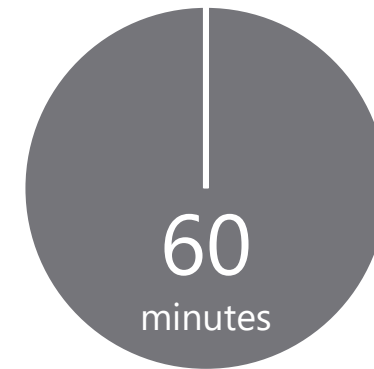
In this lab, you'll see how easy it is to set up Azure Pipelines with your GitHub projects and how you can start seeing benefits immediately.

Objectives:

- Install Azure Pipelines from the GitHub Marketplace
- Integrate a GitHub project with an Azure DevOps pipeline
- Track pull requests through the pipeline



Duration:



We're skipping this.
Can do outside class time

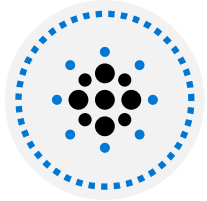
Lesson 08: Module review and takeaways



What did you learn?



Explain why continuous integration matters.



Implement continuous integration using Azure DevOps.

Module review questions

1

Name the four pillars of continuous integration.

2

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?

3

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